```java
import java.util.Scanner;

/**
 * Created by Sophey on 11/26/16.
 */
public class Mancala {

    /**
     * Moves starting with given index.
     * @param index index of the starting piece
     */
    static MancalaBoard move(int index, MancalaBoard m) {
        boolean p1Turn = m.getP1Turn();
        int[] board = new int[12];
        System.arraycopy(m.getBoard(), 0, board, 0, 12);
        int p1 = m.getP1();
        int p2 = m.getP2();
        if (p1Turn && index > 5) {
            return m;
        } else if (!p1Turn && index < 6){
            return m;
        }
        if (board[index] == 0) {
            return m;
        }
        int space = (index + 1) % 12;
        boolean lastStore = false;
        // while there are still seeds, move
        while (board[index] != 0) {
            board[index]--;
            if (!lastStore && p1Turn && space == 6) {
                p1++;
                lastStore = true;
            } else if (!lastStore && !p1Turn && space == 0) {
                p2++;
                lastStore = true;
            } else {
                board[space]++;
                space = (space + 1) % 12;
                lastStore = false;
            }
        }
        // checks if dropped in your own store on last move, do not go past
        if (lastStore) {
            return new MancalaBoard(board, p1, p2, p1Turn);
        }
        // set to last space
        space = Math.floorMod(space - 1, 12);
        // checks if dropped in an empty hole on your side, and there are pieces across
        if (p1Turn && space < 6 && board[space] == 1 && board[11 - space] > 0) {
            p1 += board[11 - space] + 1;
            board[11 - space] = 0;
            board[space] = 0;
        } else if (!p1Turn && space > 5 && board[space] == 1 && board[11 - space] > 0) {
            p2 += board[11 - space] + 1;
            board[11 - space] = 0;
            board[space] = 0;
        }
        // set to next player’s move
        p1Turn = !p1Turn;
        return new MancalaBoard(board, p1, p2, p1Turn);
    }

    /**
     * MiniMax with p1 as max
     * @param m MancalaBoard
     * @return minimax value
     */
    static int miniMax(MancalaBoard m, int depth) {
        if (depth == 0 || m.isGameOver()) {
```

```java
            return m.getWinner();
        }
        if (m.getP1Turn()) {
            int bestValue = Integer.MIN_VALUE;
            for (int i = 0; i < 6; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = miniMax(newBoard, depth - 1);
                    bestValue = Math.max(val, bestValue);
                }
            }
            return bestValue;
        } else {
            int bestValue = Integer.MAX_VALUE;
            for (int i = 6; i < 12; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = miniMax(newBoard, depth - 1);
                    bestValue = Math.min(val, bestValue);
                }
            }
            return bestValue;
        }
    }

    static int alphaBeta(MancalaBoard m, int depth, int alpha, int beta) {
        if (depth == 0 || m.isGameOver()) {
            return m.getWinner();
        }
        if (m.getP1Turn()) {
            int bestValue = Integer.MIN_VALUE;
            for (int i = 0; i < 6; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = alphaBeta(newBoard, depth - 1, alpha, beta);
                    bestValue = Math.max(val, bestValue);
                    alpha = Math.max(alpha, val);
                    if (beta <= alpha) {
                        break;
                    }
                }
            }
            return bestValue;
        } else {
            int bestValue = Integer.MAX_VALUE;
            for (int i = 6; i < 12; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = alphaBeta(newBoard, depth - 1, alpha, beta);
                    bestValue = Math.min(val, bestValue);
                    beta = Math.min(beta, val);
                    if (beta <= alpha) {
                        break;
                    }
                }
            }
            return bestValue;
        }
    }

    /**
     * Gets ideal move using minimax
     * @param m MancalaBoard
     * @return best move value
     */
    static int chooseMove(MancalaBoard m) {
        // p1’s turn
        if (m.getP1Turn()) {
            int bestValue = Integer.MIN_VALUE;
            int max = 0;
            for (int i = 0; i < 6; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = alphaBeta(newBoard, 10, Integer.MIN_VALUE, Integer.MAX_VALU
E);
                    if (val > bestValue) {
                        max = i;
                        bestValue = val;
                    }
                }
            }
            return max;
        } else {
            int bestValue = Integer.MAX_VALUE;
            int min = 0;
            for (int i = 6; i < 12; i++) {
                MancalaBoard newBoard = move(i, m);
                if (!newBoard.equals(m)) {
                    int val = alphaBeta(newBoard, 10, Integer.MIN_VALUE, Integer.MAX_VALU
E);
                    if (val < bestValue) {
                        min = i;
                        bestValue = val;
                    }
                }
            }
            return min;
        }
    }

    public static void main(String[] args) {
        MancalaBoard game = new MancalaBoard();
        Scanner s = new Scanner(System.in);
        System.out.print("Do you want to be player 1 or 2? ");
        int player = s.nextInt();
        while(!game.isGameOver()) {
            game.printBoard();
            if ((player == 1 && game.getP1Turn()) || (player == 2 && !game.getP1Turn()))
{
                System.out.println("Your move: ");
                int move = s.nextInt();
                MancalaBoard next = Mancala.move(move, game);
                if (game.equals(next)) {
                    System.out.println("Cannot make that move, choose another spot.");
                } else {
                    game = next;
                }
            } else {
                int move = Mancala.chooseMove(game);
                System.out.println(move);
                game = move(move, game);
            }
        }
        game.printBoard();
    }

}
```

```java
import java.util.Scanner;

public class MancalaBoard {

    private int[] board; // 0-5 on p1 side, 6-11 on p2
    private int p1, p2;
    private boolean p1Turn;

    public MancalaBoard() {
        board = new int[12];
        // set all values to 4
        for (int i = 0; i < board.length; i++) {
            board[i] = 4;
        }

        // each player has 0 stones
        p1 = 0;
        p2 = 0;

        p1Turn = true;
    }

    public MancalaBoard(int[] board, int p1, int p2, boolean p1Turn) {
        this.board = board;
        this.p1 = p1;
        this.p2 = p2;
        this.p1Turn = p1Turn;
    }

    public int[] getBoard() {
        return board;
    }

    public int getP1() {
        return p1;
    }

    public int getP2() {
        return p2;
    }

    public boolean getP1Turn() {
        return p1Turn;
    }

    /**
     * Checks if game is over. Returns true if is, false if not.
     * If game is over, give correct player the remaining pieces.
     */
    boolean isGameOver() {
        int index = 0;
        while (index < 6 && board[index] == 0) {
            index++;
        }
        if (index == 6) {
            for (int i = 6; i < board.length; i++) {
                p2 += board[i];
            }
            return true;
        }
        // check other side
        index = 6;
        while (index < board.length && board[index] == 0) {
            index++;
        }
        if (index == board.length) {
            for (int i = 0; i < 6; i++) {
                p1 += board[i];
            }
            return true;
```

```java
        }
        return false;
    }

    /**
     * Gets the differential between p1 and p2.
     * @return differential between p1 and p2
     */
    int getWinner() {
        return p1 - p2;
    }

    void printBoard() {
        for (int i = 11; i > 5; i--) {
            System.out.print(board[i] + "\t");
        }
        System.out.println();
        for (int i = 0; i < 6; i++) {
            System.out.print(board[i] + "\t");
        }
        System.out.println();
        System.out.println("P1: " + p1);
        System.out.println("P2: " + p2);
        System.out.println("P1 move: " + p1Turn);
    }

}
```

```java
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;
import com.sun.net.httpserver.HttpExchange;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.InetSocketAddress;

import java.util.*;

/**
 * A Mancala web service.
 *
 * @author Sophey Dong
 * @version 0.1 2016-12-13
 */

public class MancalaService {
    public static void main(String[] args) {
        // default port
        int port = 9090;

        // parse command line arguments to override defaults
        if (args.length > 0) {
            try {
                port = Integer.parseInt(args[0]);

            } catch (NumberFormatException ex) {
                System.err.println("USAGE: java MancalaService [port]");
                System.exit(1);
            }
        }

        // set up an HTTP server to listen on the selected port
        try {
            InetSocketAddress addr = new InetSocketAddress("localhost", port);
            HttpServer server = HttpServer.create(addr, 1);

            server.createContext("/move.html", new MoveHandler());
            server.createContext("/hint.html", new HintHandler());

            server.start();
        } catch (IOException ex) {
            ex.printStackTrace(System.err);
            System.err.println("Could not start server");
        }
    }

    static MancalaBoard stringToMancalaBoard(String str) {
        StringTokenizer tok = new StringTokenizer(str, ";");
        String mancala = tok.nextToken();
        String[] stringBoard = mancala.split(",");

        try {
            // convert to int board
            int[] board = new int[stringBoard.length];
            for (int i = 0; i < stringBoard.length; i++) {
                board[i] = Integer.parseInt(stringBoard[i]);
            }

            int p1 = Integer.parseInt(tok.nextToken());
            int p2 = Integer.parseInt(tok.nextToken());
            boolean p1Turn = Boolean.parseBoolean(tok.nextToken());

            return new MancalaBoard(board, p1, p2, p1Turn);

        } catch (NumberFormatException e) {
            System.err.println("error parsing");
        }
        return null;
```

```java
    }

    /**
     * An HTTP handler for roll requests.
     */
    public static class MoveHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange ex) throws IOException {
            // we assume the query encodes the current state as n, n, n;p1;p2;p1turn;move
            // where n represents the number of stones in the pile at that index
            // p1 & p2 represent the number of stones in each mancala
            // p1turn is true or false, depending on which player's turn it is
            // move is the bucket the player chose

            System.err.println(ex.getRequestURI());
            String q = ex.getRequestURI().getQuery();

            StringBuilder reponse = new StringBuilder();

            // decode the string
            StringTokenizer tok = new StringTokenizer(q, ";");
            if (tok.countTokens() != 5) {
                sendResponse(ex, error(q, "malformed state"));
            } else {
                MancalaBoard mancalaBoard = MancalaService.stringToMancalaBoard(q);
                int move = Integer.parseInt(q.substring(q.lastIndexOf(";") + 1));

                boolean p1Turn = mancalaBoard.getP1Turn();

                if ((!p1Turn && move > 5) || (p1Turn && move < 6)) {
                    sendResponse(ex, error(q, "selected wrong side"));
                } else {
                    // move is legal!

                    if (p1Turn) {
                        move -= 6;
                    } else {
                        move = 11 - move;
                    }
                    mancalaBoard = Mancala.move(move, mancalaBoard);

                    boolean isGameOver = mancalaBoard.isGameOver();

                    StringBuilder newState = new StringBuilder();
                    if (isGameOver) {
                        for (int i = 0; i < 12; i++) {
                            newState.append("0,");
                        }
                    } else {
                        for (int n : mancalaBoard.getBoard()) {
                            newState.append(n + ",");
                        }
                    }
                    // build the response object
                    Map<String, String> response = new HashMap<String, String>();
                    response.put("state", q);
                    response.put("mancala", newState.toString());
                    response.put("turn", "" + (mancalaBoard.getP1Turn() ? 1 : 2));
                    response.put("p1", "" + mancalaBoard.getP1());
                    response.put("p2", "" + mancalaBoard.getP2());
                    response.put("over", "" + isGameOver);
                    response.put("winner", (mancalaBoard.getWinner() > 0 ? "1" : "2"));

                    sendResponse(ex, response);
                }
            }
        }
    }

    /**
```

```
     * An HTTP handler for hint requests.
     */
    public static class HintHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange ex) throws IOException {
            System.err.println(ex.getRequestURI());
            String q = ex.getRequestURI().getQuery();
            StringTokenizer tok = new StringTokenizer(q, ";");

            MancalaBoard mancalaBoard = MancalaService.stringToMancalaBoard(q);

            if (mancalaBoard == null) {
                error(q, "not correct string");
            }

            int hint = Mancala.chooseMove(mancalaBoard);
            if (mancalaBoard.getP1Turn())
                hint = hint + 6;
            else
                hint = 11 - hint;

            Map<String, String> response = new HashMap<String, String>();
            response.put("state", q);
            response.put("hint", "" + hint);

            sendResponse(ex, response);
        }
    }

    /**
     * Returns a map containing key-value pairs for the given state and message.
     *
     * @param state   a string
     * @param message a string
     * @return a map containing key-value pairs for state and message
     */
    private static Map<String, String> error(String state, String message) {
        Map<String, String> result = new HashMap<String, String>();

        result.put("state", state);
        result.put("message", message);

        return result;
    }

    /**
     * Sends a JSON object as a response in the given HTTP exchange.  Each key-value pair
     * in the given map will be copied to the JSON object.
     *
     * @param ex   an HTTP exchange
     * @param info a non-empty map
     */
    private static void sendResponse(HttpExchange ex, Map<String, String> info) throws IO
Exception {
        // write the response as JSON
        StringBuilder response = new StringBuilder("{");
        for (Map.Entry<String, String> e : info.entrySet()) {
            response.append("\"").append(e.getKey()).append("\":")
                    .append("\"").append(e.getValue()).append("\",");
        }
        response.deleteCharAt(response.length() - 1); // remove last ,
        response.append("}"); // close JSON

        ex.getResponseHeaders().add("Access-Control-Allow-Origin", "*");
        byte[] responseBytes = response.toString().getBytes();
        ex.sendResponseHeaders(HttpURLConnection.HTTP_OK, responseBytes.length);
        ex.getResponseBody().write(responseBytes);
        ex.close();
    }
}
```

```
#wholedisplay {
    float: left;
}
#display {
    height: 50px;
    padding-right: 1em;
    float: left;
    position: relative;
}

.mancala {
    display: inline;
    font-family: Verdana, sans-serif;
    font-weight: bold;
    font-size: 30px;
    padding-right: 1.5em;
}

#p2D {
    float: left;
    padding-right: 2em;
    font-family: Verdana, sans-serif;
    font-weight: bold;
}

.pM {
    color: orange;
    font-size: 55px;
}

#p1D {
    float: left;
    font-family: Verdana, sans-serif;
    font-weight: bold;
}

#buttons {
    float: left;
    clear: left;
}

[data-selected=true]
{
    color: blue;
}
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">

<HTML>
<HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=iso-8859-1">
    <TITLE>Mancala</TITLE>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=iso-8859-1">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></scri
pt>
    <script src="mancala.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="mancala.css">
</HEAD>


<div id="wholedisplay">
    <div id="p2D"><div class="pM" id="p2">0</div><p style="font-size: 10px; color: lights
kyblue;">Player 2</div>
    <div id="display">
        <div id="mancala0" class="mancala" data-selected="false">4</div>
    </div>
    <div id="p1D"><div class="pM" id="p1">0</div><p style="font-size: 10px; color: lights
kyblue">Player 1</div>
</div>
<div id="buttons" style="text-align:left; margin:auto">
    <br><br>
    <button type="button" onclick="moveStones()">Move</button>
    <button type="button" onclick="hint()">Hint</button>
    <p>
        Player <span id="player">1's Turn</span>
</div>
<span id="message">&nbsp;</span>


<script type="text/javascript">
    $(document).ready(init);

    function init() {
        initMancala();
    }
</script>

</BODY>
</HTML>
```

```
// sets up the HTML elements that display the mancala board
function initMancala() {
    makeMancala();

    // initialize global variables to keep track of game state
    board = ""; // board representation
    turn = 1;      // the current player
    p1 = 0;
    p2 = 0;
    move = -1;
    state = "";
}

// makes the HTML elements to hold the board
function makeMancala() {
    var display = $("#display");
    var p1D = $("")
    var m = $("#mancala0");
    for (var i = 10; i > 5; i--) {
        display.append(m.clone());
    }
    display.append("<br>");
    for (var i = 0; i < 6; i++) {
        display.append(m.clone());
    }

    $(".mancala").click(selectMancala);
}

function selectMancala() {
    if ($(this).attr('data-selected') == 'false') {
        $(".mancala").each(function () {
            $(this).attr('data-selected', 'false');
        });
        $(this).attr('data-selected', 'true');
    }
    else {
        $(this).attr('data-selected', 'false');
    }
}

function isMove(elt) {
    if ($(elt).attr('data-selected') == 'true') {
        return true;
    }
    else {
        return false;
    }
}

function encodeState() {
    var mArray = new Array(12);
    $(".mancala").each(function (i) {
        if (i < 6)
            mArray[11 - i] = $(this).text();
        else
            mArray[i - 6] = $(this).text();
        if (isMove(this))
            move = i;
    });
    var mancala = "";
    mArray.forEach(function (val) {
        mancala += val + ",";
    });
    state = mancala + ";" + p1 + ";" + p2 + ";" + (turn == 1 ? true : false) + ";" + move
;
    console.log(state);
    return state;
}
```

```
function moveStones() {
    state = encodeState();
    $.getJSON("http://localhost:9090/move.html?" + state, doMove);
}

function hint() {
    state = encodeState();
    $.getJSON("http://localhost:9090/hint.html?" + state, showHint);
}

function showHint(result) {
    console.log(result);
    if (result.state == state) {
        if ("message" in result) {
            $("#message").text(result.message);
        }
        else {
            $(".mancala").each(function (i) {
                if (i == result.hint)
                    $(this).attr('data-selected', 'true');
                else
                    $(this).attr('data-selected', 'false');
            });
        }
    }
}

function doMove(result) {
    console.log(result);
    if (result.state == state) {
        if ("message" in result) {
            $("#message").text(result.message);
        }
        else {
            $("#message").text("");
            state = result.mancala + ";" + result.p1 + ";" + result.p2 + ";" + result.tur
n;
            turn = result.turn;
            //$("#player").text(turn);
            if (result.over == "true") {
                $("#player").text(result.winner + " wins!");
            }
            else {
                $("#player").text(turn + "'s turn");
            }
            var board = result.mancala.split(',');
            $(".mancala").each(function (i) {
                $(this).attr('data-selected', 'false');
                if (i < 6)
                    $(this).text(board[11 - i]);
                else
                    $(this).text(board[i - 6]);
            });
            p1 = result.p1;
            p2 = result.p2;
            $('#p1').text(result.p1);
            $('#p2').text(result.p2);
        }
    }
}
```