Webシステム基礎実験レポート

(ポーカーアプリ)　2132023 内山裕太

1. 概要

今回作成したポーカーアプリの遊び方について説明する．

（URL… <http://localhost:8080/s2132023/PokerServlet> ）

URLをクリックするとゲームが開始される．初期画面の表示内容について上部から説明する．

・ポーカーゲーム…ゲームの名称である．

・ゲーム回数　　…現在のゲーム回数を表示している．

　　リセットすることで１に戻る．

・チップ　　　　…現在のチップの残り状況を表示する．

　 成立役に応じて増減する．

次にゲームのルール，進行，操作方法について説明する．

初めに，初期の手札の中から交換したいカードの画像下部に表示されているチェックボックスにチェックを入れる．次に，交換ボタンをクリックするとカードが交換され，新しいカードが山札から配られる．その時点での成立役でチップを獲得できる．ゲームに参加するには

100枚のチップが必要であり，チップが0を下回ると

ゲームオーバーとなる．参加した時点でチップがマイナスになっても，成立役などでチップをプラスに戻すことができれば，ゲームは継続することができる．

最下部にあるリセットボタンをクリックすることで，いつでもゲームをリスタートできる．自分の好きな手札から始めることができる．ゲームに関する説明は以上である．

1. 動作

アプリケーションの内部処理について説明する．

1. ゲームを開始するとPokerServletクラス(以下Servletクラス)のPokerServletメソッドの実行が始まり，PokerModelクラス(以下Modelクラス)のインスタンスを生成する．ここでコンストラクタが実行され，手札と山札のリストを宣言している．

PokerModel model;

    public PokerServlet() {

        super();

        //PokerModelクラスのインスタンス生成，コンストラクタ実行

        model = new PokerModel();

    }

public PokerModel() {

        //手札と山札のListを生成

        deckcards = new ArrayList<>();

        handcards = new ArrayList<>();

    }

1. 次にdoGetメソッドが実行される．このメソッドは，ゲーム開始後，Modelクラスのコンストラクタ実行後に呼び出されるメソッドである．ここではModelクラスのresetメソッドとnextgameメソッドを呼び出している．以下に各メソッドの説明を記載する．
   1. resetメソッド

所持チップ枚数とゲーム数を初期化するメソッド．

それぞれの初期値は，チップが５００枚，ゲーム数が１になるようにしている．（ゲーム参加に１００枚消費するため，表示上は４００枚になっている）

* 1. nextgameメソッド

ゲームをスタートするための準備を行うメソッド．

* + - 参加費分のチップを所有チップから減らす．
    - 山札を構築する．
    - 手札を構築する．

主に上記の３点を行う．

最後にpoker.jsp内でModelクラス内のメソッドや変数にアクセスするための処理を行っている．

protected void doGet(HttpServletRequest request, HttpServletResponse response)

            throws ServletException, IOException {

        model.reset();

        model.nextgame();

        //jspファイルからmodelの値を受け取る準備

        request.setAttribute("model", model);

        //jsp側でページ表示を行う

        RequestDispatcher dispatcher = request.getRequestDispatcher("/poker.jsp");

        dispatcher.forward(request, response);

    }

1. 次に，doGetメソッドで生成した情報をもとに

poker.jspファイルでゲーム画面を生成する．

* ゲーム回数
* 残りチップ数
* 参加費
* 初期の手札

上記４点が表示される．

1. 最後に，doPostメソッドが実行される．このメソッドは，交換ボタンが押されると同時に呼び出されるメソッドである．ここでは，交換対象を選んだ結果に応じた処理を行う．場合分けについては，以下に記載する．
   1. 交換対象を選択しない場合

交換対象を選択しない場合は，Modelクラスのchangeメソッドに空のArrayList型リストであるchangelistを渡す．

* 1. 交換対象を選択した場合

交換対象を選択した場合は，選択した手札の要素番号を配列change[]に与える．

同じ内容のArrayList型リストを作成し，change[]の内容をコピーする．作成したArrayList型リストをModelクラスのchangeメソッドに与える．

changeメソッドでは，changelistを受け取り，要素と手札の要素数が一致するものを山札の最上部と交換する処理を行う．また，交換処理が終わった後に役の判別を行う．結果によってチップが０以下になった場合は再度ゲームが始まる．チップが残っている限りゲームは続く仕様である．

protected void doPost(HttpServletRequest request, HttpServletResponse response)

            throws ServletException, IOException {

        request.setCharacterEncoding("UTF-8");

        response.setContentType("text/html: charset=UTF-8");

        //交換ボタンを押された時の処理

        if(model.getButtonLabel().equals("交換")) {

            //交換対象に選択されたカード情報を格納

            String change[] = request.getParameterValues("change");

            //空のArrayListを生成

            List<String>changelist = null;

            //交換しない場合

            if(change == null) {

                changelist = new ArrayList<>();

            //交換を行う場合

            } else {

                changelist = Arrays.asList(change);

            }

            //手札交換の実行

            model.change(changelist);

//次のゲームを押された時の処理

        } else if(model.getChips() > 0) {

            model.nextgame();

        //リセット，ゲームオーバーの場合の処理

        } else {

            model.reset();

            model.nextgame();

        }

        //jspファイルからmodelの値を受け取る準備

        request.setAttribute("model", model);

        //jsp側でページ表示を行う準備

        RequestDispatcher dispatcher = request.getRequestDispatcher("/poker.jsp");

        dispatcher.forward(request,  response);

    }

以下に役の判別法を記載する．

* ロイヤルストレートフラッシュ

専用のcountLoyalメソッド，countFlashメソッドを作成し，判定している．

* countLoyalメソッド

手札のカード内で，A，10，J，Q，Kがそろっているかを判断している．その際に，countNumberメソッドを用いているが，このメソッドは手札の数字ごとの枚数を管理するメソッドである．

//ロイヤルストレートフラッシュの判断

    boolean countLoyal() {

        boolean loyal = false;

        if(count[0]==1 && count[9]==1 && count[10]==1 && count[11]==1 && count[12]==1) {

            loyal = true;

        }

        return loyal;

    }

* countFlashメソッド

トランプは各図柄１３枚ずつであるため，手札の各数字を１３で割った商で図柄を判定している．１枚目の手札の数字を１３で割った商を

変数checkに代入し，すべての商がcheckに一致するかを確認している．

//フラッシュの判断

    boolean countFlash() {

        boolean flash = false;

        int check;

        if(handcards.get(0)%13 == 0) {

            check = handcards.get(0) / 13 - 1;

        }else {

            check = handcards.get(0) / 13;

        }

        for(int i=1; i<5; i++) {

            if(handcards.get(i)/13==check || handcards.get(i)==13\*(check+1)) {

                if(i==4) {

                    flash = true;

                }

            }else {

                break;

            }

        }

        return flash;

    }

* ストレートフラッシュ，ストレート

専用のstraightメソッドと上記のflashメソッドを用いて判定する．

* countStraightメソッド

手札の数字の最大値と最小値の差が４であること，数字の重複がないことを基準として数字が並んでいるかを判定するメソッド．

//ストレートの判断

    boolean countStraight() {

        boolean straight = false;

        int max = handcards.get(0);

        int min = handcards.get(0);

        int check;

        for(int i=1; i<5; i++) {

            check = handcards.get(i) % 13;

            if(check==0) {

                max = 13;

            }else if(check==1) {

                min = 1;

            }else if(max < check) {

                max = check;

            }else if(min > check) {

                min = check;

            }

        }

        if(max-min==4) {

            straight = true;

        }

        for(int i: count) {

            if(i>=2) {

                straight = false;

                break;

            }

        }

        return straight;

    }

* フォーカード

専用のfourメソッドを用いて判定する．

* fourメソッド

countNumberメソッドで要素に４が含まれているかを基準として判定するメソッド．

//フォーカードの判断

    boolean countFour() {

        boolean four = false;

        for(int i: count) {

            if(i==4) {

                four = true;

            }

        }

        return four;

    }

* スリーカード

専用のthreeメソッドを用いて判定する．

* threeメソッド

countNumberメソッドで要素に３が含まれているかを基準として判定するメソッド．

//スリーカードの判断

    boolean countThree(){

        boolean three = false;

        for(int i: count) {

            if(i==3) {

                three = true;

            }

        }

        return three;

    }

* ワンペア，ツーペア

countNumberメソッドで要素に２がいくつ含まれているかを基準として判定する．

//ペアの数を数えるメソッド

    int countPair() {

        int pair = 0;

        for(int i: count) {

            if(i==2) {

                pair += 1;

            }

        }

        return pair;

    }

* フルハウス

スリーカードとワンペアの同時成立で判定する．

また，獲得チップ数は最高ティアのロイヤルストレートフラッシュを５００枚とし，成立役がワンランク下がるにつれて，－５０枚としている．

//強い役から判別

        if(loyal==true && flash==true) {

            message = "ロイヤルストレートフラッシュ";

            point = 500;

        }else if(straight==true&&flash==true) {

            message = "ストレートフラッシュ";

            point = 450;

        }else if(four==true) {

            message = "フォーカード";

            point = 400;

        }else if(three==true && pair==1) {

            message = "フルハウス";

            point = 350;

        }else if(flash==true) {

            message = "フラッシュ";

            point = 300;

        }else if(straight==true) {

            message = "ストレート";

            point = 250;

        }else if(three==true) {

            message = "スリーカード";

            point = 200;

        }else if(pair==2) {

            message = "ツーペア";

            point = 150;

        }else if(pair==1) {

            message = "ワンペア";

            point = 100;

        }else if(seven > 0) {

            message = "セブン";

            point = seven \* 10;

        }else {

            message = "ハイカード";

            point = -200;

        }

以下はpoker.jspファイルである．

<%@ page language="java" contentType="text/html; charset=UTF-8"

    pageEncoding="UTF-8"%>

<%@ page import="poker.PokerModel"%>

<%

PokerModel model = (PokerModel)request.getAttribute("model");

String label = model.getButtonLabel();

%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Poker</title>

</head>

<body>

ポーカーゲーム

<hr>

ゲーム回数：<%= model.getGames() %>

<br>

チップ：<%= model.getChips() %>(参加費済)

<br>

参加費：<%= model.getChallenge() %>

<hr>

<%= model.getMessage() %>

<form action="/s2132023/PokerServlet" method="POST">

<table>

<tr>

<td><img src="cards/<%= model.getHandcardAt(0) %>.png" width="100" height="150"></td>

<td><img src="cards/<%= model.getHandcardAt(1) %>.png" width="100" height="150"></td>

<td><img src="cards/<%= model.getHandcardAt(2) %>.png" width="100" height="150"></td>

<td><img src="cards/<%= model.getHandcardAt(3) %>.png" width="100" height="150"></td>

<td><img src="cards/<%= model.getHandcardAt(4) %>.png" width="100" height="150"></td>

</tr>

<tr align="center">

<td><input type="checkbox" name="change" value="0"></td>

<td><input type="checkbox" name="change" value="1"></td>

<td><input type="checkbox" name="change" value="2"></td>

<td><input type="checkbox" name="change" value="3"></td>

<td><input type="checkbox" name="change" value="4"></td>

</tr>

</table>

<input type="submit" value="<%= label %>">

</form>

<hr>

<a href="/s2132023/PokerServlet">リセット</a>

</body>

</html>

* 1. 考察

1. 今回のプログラム作成において，１つ１つの役を判定するためのメソッドを作成したが，１番大きく感じたメリットとして修正の容易さがあげられる．１つの判定メソッドが完成するたびに動作確認をし，その都度不具合を修正することができる．今回は規模のかなり小さなプログラムであるが，規模が大きくなるにつれメソッド作成の重要度は上がると考えられる．
2. メソッドを作成する際に，標準のライブラリを利用することは極めて重要である．処理内容を考え，実際に記述するには多くの時間を要するため，似た処理を行うライブラリが存在するかを調査し，存在する場合はコード作成の参考にすることで大きく時間を短縮できるだろう．
3. 課題にない機能の実装箇所を以下に記載する．
4. ゲームを開始する際に，参加料としてチップを１００枚消費するようにした．これは，ゲームが無限に終了しなくなる可能性を考慮したためである．今回のプログラムには「セブン」という役が存在するため，終了しなくなる可能性はかなり高いと考えられる．
   * + 1. ゲームオーバーになってしまった際に，何ゲーム継続することができたかを表示するようにした．
       2. 実際のポーカーと同等の役の種類を実装した．