Swingによるデスクトップアプリケーション開発（ポーカー）-JavaSE1.8

office・M
2024年10月13日 14:33

¥1,000 •••

Java8のSwing環境でデスクトップアプリケーションの開発方法を学ぶ講座をシリーズで提供しています。今回はWindowBuilder（Swingデザイナー）を使ってカードゲームのポーカーを作成します。

2024年9月よりECLIPSEのバージョンを最新版（Version: 2024-06 (4.32.0)）に変更しました。

目次

外部設計

WindowBuilderのSwingデザイナーで図１のようなGUIを作成します。最初は10枚のカード中5枚が表向きに配布され、その時点での役が表示されています。最初の持ち点は100で掛け点は10です。1回だけカードの交換が可能です。表向き裏向きにかかわらず残すカードをチェックして交換ボタンをクリックすることにより図２のように役が確定し配点が計算されます。

図１．最初の10枚のカード配布と残すカードのチェック

図2．役の確定と配点計算

カードの画像は事前に用意しておきます。今回は60×90ピクセルの52枚分のカード画像を用意しました。

カードのファイルは３桁の番号で作成されています。百の位の数字がカードの種類を表しており以下のように設定します。

クラブ：101～113　ダイヤ：201～213　ハート：301～313
スペード：401～413

内部設計

処理ロジック

以下の条件を実現するPokerアプリケーションの作成を行います。
（条件）
・アプリを起動すると52枚のカードをシャッフルし10枚取り出します。５枚は表向きで、残りの5枚は裏向きで表示します。これを初期配布とします。
・初期配布の状態で表向き５枚の役を判定し表示します。また持ち点と掛け点のテキストボックス、カード交換のチェックボックス及びカード交換のボタンを表示します。
・カード構成は１回だけ交換可能です。役の状態を判断して表向きの残したいカードと裏向きの残したいカードをチェックボックスで指定します。
・「カード交換ボタン」をクリックすると残したカードの役が確定され払い戻しの点数が計算されます。
・役の倍率や処理の流れのイメージは外部設計の図1や図2を参照してください。

提供クラス

・CardShuffleArrayListBean.class

（仕様）
52枚のカードをシャッフルします。結果はArrayListクラスに保存します。
（使い方）
・引数なしのコンストラクタで実体化します。
・shuffleメソッドを実行することでカードの構成がシャッフルされ配列（ArrayList）に保持されます。
・利用側のクラスはgetListメソッドで配列の中身を取得します。

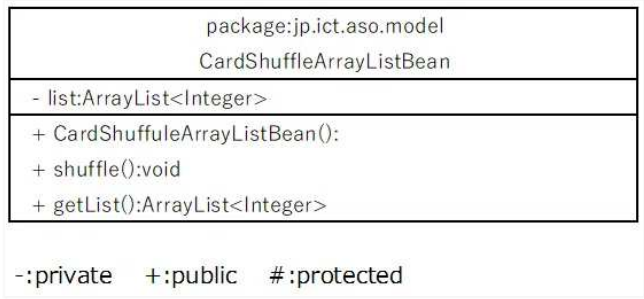

図3．CardShuffleArrayListBeanクラス図

・CardShuffleHtml10Bean.class

（仕様）
シャッフル済みの52枚のカードの最初から10枚を取り出し5枚は表向きで残りの5枚は裏向きに設定します。**チェックボックスは付きません。**
（使い方）
・引数なしのコンストラクタで実体化します。
・setListメソッドにカード構成の配列（ArrayList）を設定すると最初の10枚のカードがhtml文字列でフィールド内に表組されます。
・最初の5枚は表向きで、後の5枚は裏向きで設定されます。
・利用側のクラスはgetShuffleCardImage10メソッドでフィールドの情報を取得します。

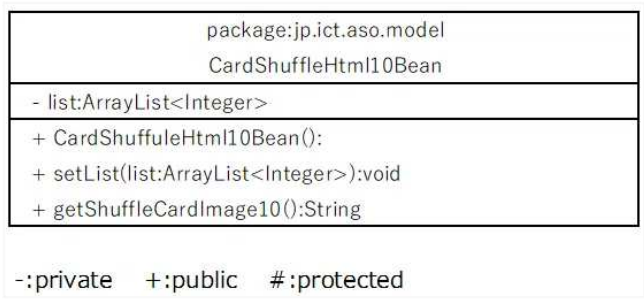

図４．CardShuffleHtml10Beanクラス図

・CardHanteiHtml5Bean.class

（仕様）
初期配布状態の表向きの5枚のカードの役を**CardScoreBeanを用いて判定**し設定します。
（使い方）
・引数なしのコンストラクタで実体化します。
・setListメソッドに10枚のカード構成の配列（ArrayList）を設定します。
・hanteiCard5メソッドの実行で最初の5枚のカード構成の役を判定します。
・役の判定にはCardScoreBeanクラスを用います。
・利用側のクラスはgetHantei5メソッドで役を取得します。

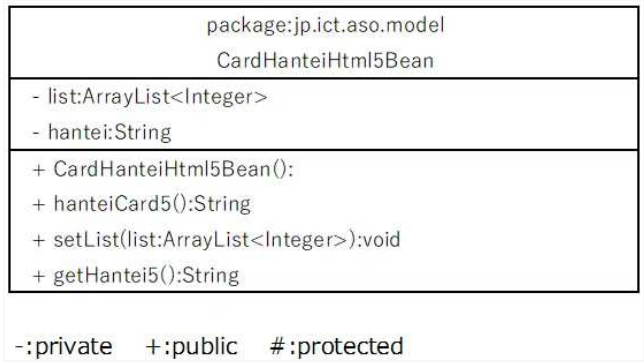

図５．CardHanteiHtml5Beanクラス図

・CardConfirmHtml5Bean.class

（仕様）
チェックされた5枚のカードの役を**CardScoreBeanを用いて判定**し設定します。

（使い方）
・引数なしのコンストラクタで実体化します。
・setCheckedCardメソッドにチェック済みの5枚のカード構成の配列を設定します。
・setKaketenメソッドで掛け点を設定します。
・hanteiCard5メソッドの実行で5枚のカード構成の役を判定します。
・役の判定と払い戻し点数の計算にはCardScoreBeanクラスを用います。
・利用側のクラスはgetHantei5メソッドで役を取得します。

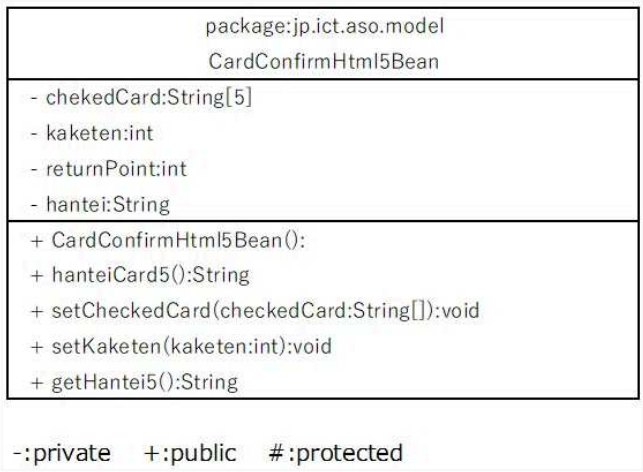

図６．CardConfirmHtml5Beanクラス図

・CardScoreBean.class

（仕様）
5枚のカードの役を判定し払い戻し点数を計算します。役と払い戻しの倍率は以下のようになります。
・ワンペア１倍、ツーペア２倍、スリーカード５倍、フラッシュ６倍、ストレート６倍、フォーカード８倍、フルハウス10倍
（使い方）
・引数なしのコンストラクタで実体化します。
・setCardメソッドの1～５にカードの数値情報を設定します。
・setKaketenメソッドで掛け点を設定します。
・scoreメソッドの実行で5枚のカード構成の役の判定と払い戻し点数を計算します。
・利用側のクラスはgetMessageメソッドで役を取得し、getReturnPointメソッドで払い戻し点数を取得します。

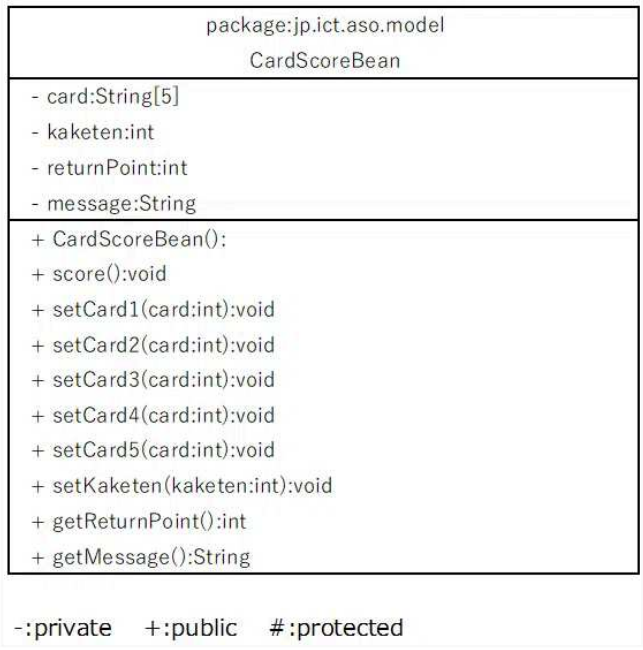

図7．CardScoreBeanクラス図

実装準備

プロジェクトの作成

Eclipseのメニューバーより
ファイル→新規→Javaプロジェクト→「SwingPoker」プロジェクトを作成
→図８の内容で設定する

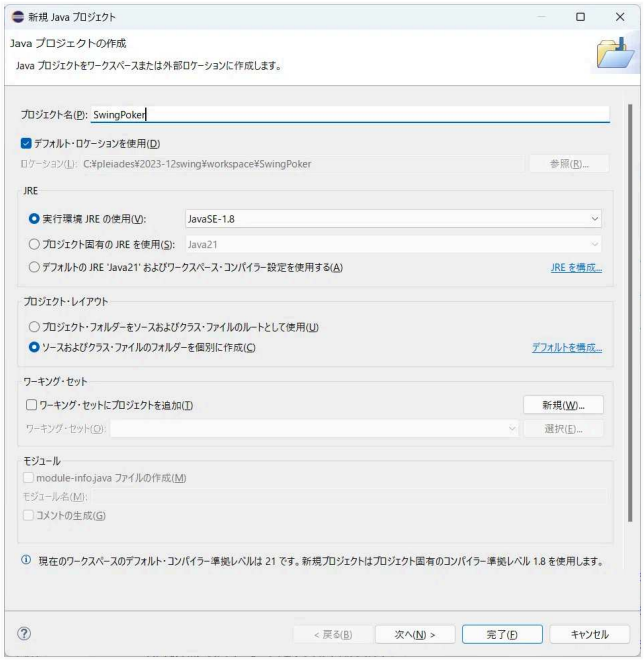

図8．SwingPokerプロジェクト設定

※作成済みであればこの処理は必要ありません。

以下画面のスクリーンショットはライトテーマで取得します。
（ライトテーマの設定方法）
Eclipseのメニューバーより
　ウィンドウ → 設定 → 一般 → 外観 → ルック＆フィール → ライト
　→ 適用して閉じる → Eclipseの再起動がかかります

外部classの利用準備

以下Windows環境を想定しています。
事前に「**JavaBeansPokerフォルダ**」を作成しておきます。
（例　c:￥JavaBeansPoker）

JavaBeansPokerフォルダ内に以下のの**５つのクラスファイル**を保存しておきます。
（CardShuffleArrayListBean.class、CardShuffleHtml10Bean.class、CardHanteiHtml5Bean.class、CardConfirmHtml5Bean.class、CardScoreBean.class）

その際**パッケージの階層**に従ってください。
（例　c:￥JavaBeansPoker￥jp￥ict￥aso￥model￥〇〇〇.class）

ビルド・パスに追加

５つのクラスファイルをビルド・パスに追加します。

Eclipseパッケージ・エクスプローラより
SwingPokerプロジェクトを右クリック→ビルド・パス→ビルド・パスの構成→「ライブラリ」タブ→「クラスパス」クリック→外部クラス・フォルダーの追加→「JavaBeansPoker」を指定します→最後に「適用して閉じる」をクリック

※図９のように一度設定されていれば再度設定する必要はありません。

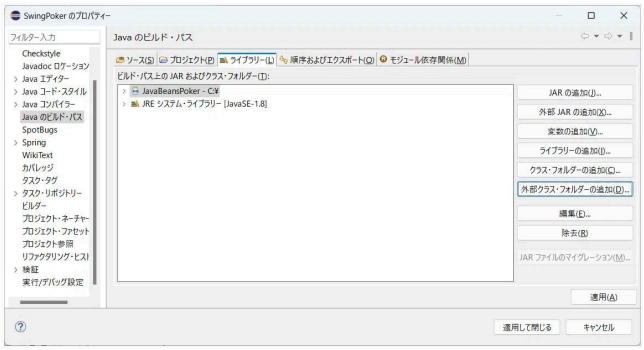

図9．外部クラス・フォルダの追加

実装

ひな形の作成

WindowBuilderを用いてSwingアプリケーションのスケルトン（骨格）を自動生成させます。

Eclipseパッケージ・エクスプローラより
SwingPokerプロジェクトを右クリック→新規→その他
→ WindowBuilder → Swingデザイナー → JFrameを選択 → 次へ

以下の内容で作成
　　パッケージ：jp.ict.aso.swing
　　名前：Poker

GUI実装

自動生成されたプログラム（スケルトン）からGUIのデザインを実装します。

パレットと構造（コンポーネント、プロパティ）のViewを利用するのがコツです。デザインイメージは設定反映の参考としてとらえた方が良いでしょう。

①画面中央下部にあるデザインタブでソースコード編集画面からSwingデザイナーに切り替えます。

②contentPaneのLayoutプロパティをBorderLayoutに設定します。

③contentPaneの「North」「South」「Center」の順番でGUI部品のJPanelをパレットから配置します。各パネルのプロパティのConstrainsは「**panel**が**North**」「**panel_1**が**South**」「**panel_2**が**Center**」となります。

④「**Center**」位置のJPanel(**panel_2**)のプロパティのLayoutを**BoxLayout**に変更し、Layoutの＋を展開しConstructorの＋を展開した**axisの設定をY_AXIS**にします。

⑤「**South**」位置のJPanel(**panel_1**)のプロパティのLayoutを**BoxLayout**に変更し、Layoutの＋を展開しConstructorの＋を展開した**axisの設定をY_AXIS**にします。

⑥「**North**」位置のJPanel(**panel**)にGUI部品のJLabelを１つ、JCheckBoxを５つ、JToggleButtonを１つ、順番にパレットから配置します。各部品のtextプロパティを図10のように変更します。
また、全てのJCheckBoxのプロパティのselectedはtrueに変更します。

⑦「**South**」位置のJPanel(**panel_1**)にGUI部品のJPanelを２つパレットから配置します（panel_3とpanel_4が縦に配置されます）。

⑧「**South**」位置のJPanel(**panel_3**)にGUI部品のJLabelを１つ、JCheckBoxを５つ、JToggleButtonを１つ、順番にパレットから配置します。各部品のtextプロパティを図10のように変更します。

⑨「**South**」位置のJPanel(**panel_4**)にGUI部品のJButtonを１つ、JLabelを２つ、JTextFieldを１つ、JLabelを１つ、JTextFieldを１つ、JLabelを１つ、JButtonを１つ、順番にパレットから配置します。各部品のtextプロパティを図10のように変更します。

⑩「**Center**」位置のJPanel(**panel_2**)にGUI部品のJEditorPaneを２つパレットから配置します（図10のようにeditorPaneとeditorPane_１が縦に配置されます）。**editorPane**が配布カードの表示領域、**editorPane_１**が役の判定結果の表示領域となりますので、それぞれのプロパティの**contentType**を**text/html**に変更してください。

ここまでのGUIの設計状況（部品配置）は図10のようになります。

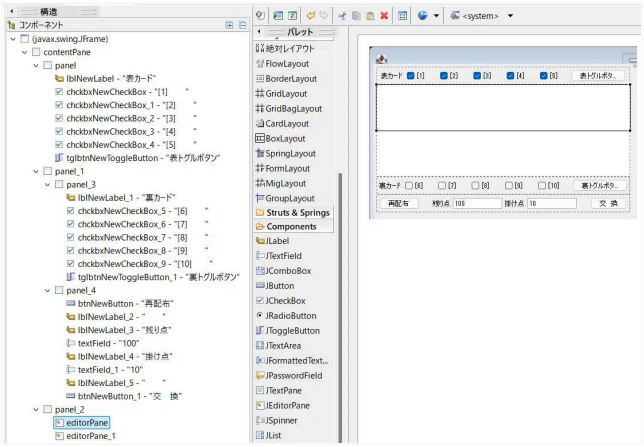

図10．部品配置

⑪ボタンにイベントリスナーを対応付けます。パレットのSwingActions内にある「新規」のリスナーを選択してボタンをクリックすることで対応付けられます。

最終的なデザインは図11のようになります。

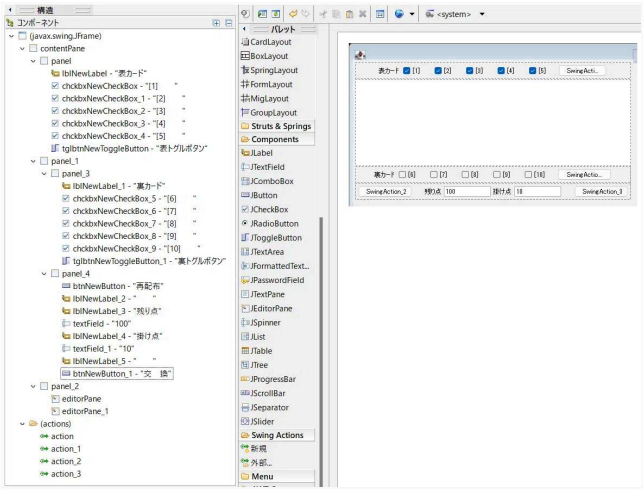

図11．最終デザイン

イベント実装

ソースタブに変更します。

①「表トグルボタン（反転ボタン）」のイベントのソース部分を変更します。

```
private class SwingAction extends AbstractAction {
    public SwingAction() {
        putValue(NAME, "反転");
        putValue(SHORT_DESCRIPTION, "チェックを反転します");
    }
    public void actionPerformed(ActionEvent e) {
        if(chckbxNewCheckBox.isSelected()) {
            chckbxNewCheckBox.setSelected(false);
        }else {
            chckbxNewCheckBox.setSelected(true);
        }
        if(chckbxNewCheckBox_1.isSelected()) {
            chckbxNewCheckBox_1.setSelected(false);
        }else {
            chckbxNewCheckBox_1.setSelected(true);
        }
        if(chckbxNewCheckBox_2.isSelected()) {
            chckbxNewCheckBox_2.setSelected(false);
        }else {
            chckbxNewCheckBox_2.setSelected(true);
        }
        if(chckbxNewCheckBox_3.isSelected()) {
            chckbxNewCheckBox_3.setSelected(false);
        }else {
            chckbxNewCheckBox_3.setSelected(true);
        }
        if(chckbxNewCheckBox_4.isSelected()) {
            chckbxNewCheckBox_4.setSelected(false);
        }else {
            chckbxNewCheckBox_4.setSelected(true);
        }
    }
}
```

図12．表トグルボタン（反転ボタン）のイベント内容

②「裏トグルボタン（反転ボタン）」のイベントのソース部分を変更します。

```
private class SwingAction_1 extends AbstractAction {
    public SwingAction_1() {
        putValue(NAME, "反転");
        putValue(SHORT_DESCRIPTION, "チェックを反転します");
    }
    public void actionPerformed(ActionEvent e) {
        if(chckbxNewCheckBox_5.isSelected()) {
            chckbxNewCheckBox_5.setSelected(false);
        }else {
            chckbxNewCheckBox_5.setSelected(true);
        }
        if(chckbxNewCheckBox_6.isSelected()) {
            chckbxNewCheckBox_6.setSelected(false);
        }else {
            chckbxNewCheckBox_6.setSelected(true);
        }
        if(chckbxNewCheckBox_7.isSelected()) {
            chckbxNewCheckBox_7.setSelected(false);
        }else {
            chckbxNewCheckBox_7.setSelected(true);
        }
        if(chckbxNewCheckBox_8.isSelected()) {
            chckbxNewCheckBox_8.setSelected(false);
        }else {
            chckbxNewCheckBox_8.setSelected(true);
        }
        if(chckbxNewCheckBox_9.isSelected()) {
            chckbxNewCheckBox_9.setSelected(false);
        }else {
            chckbxNewCheckBox_9.setSelected(true);
        }
    }
}
```

図13．裏トグルボタン（反転ボタン）のイベント内容

③「再配布ボタン」のイベントのソース部分を変更します。

```
private class SwingAction_2 extends AbstractAction {
    public SwingAction_2() {
        putValue(NAME, "再配布");
        putValue(SHORT_DESCRIPTION, "再度10枚配布します");
    }
    public void actionPerformed(ActionEvent e) {
        chckbxNewCheckBox.setSelected(true);
        chckbxNewCheckBox_1.setSelected(true);
        chckbxNewCheckBox_2.setSelected(true);
        chckbxNewCheckBox_3.setSelected(true);
        chckbxNewCheckBox_4.setSelected(true);
        chckbxNewCheckBox_5.setSelected(false);
        chckbxNewCheckBox_6.setSelected(false);
        chckbxNewCheckBox_7.setSelected(false);
        chckbxNewCheckBox_8.setSelected(false);
        chckbxNewCheckBox_9.setSelected(false);

        cardShuffle();
        btnNewButton.setEnabled(true);
    }
}
```

図14．再配布ボタンのイベント内容

④「交換ボタン」のイベントのソース部分を変更します。

```
private class SwingAction_3 extends AbstractAction {
    public SwingAction_3() {
        putValue(NAME, "交　換");
        putValue(SHORT_DESCRIPTION, "裏カードも含めて５枚選択します");
    }
    public void actionPerformed(ActionEvent e) {
        int mochiten=Integer.parseInt(textField.getText());
        int kaketen=Integer.parseInt(textField_1.getText());
        // 確定の５枚を配列に保存
            cardCheck();

        // 確定の５枚を出力して役を判定
            CardConfirmHtml5Bean cc5 = new CardConfirmHtml5Bean();
            cc5.setCheckedCard(checkedCard);
            cc5.setKaketen(kaketen);
            cc5.hanteiCard5();
            editorPane.setText(cc5.getCheckedCardImage5());
            editorPane_1.setText(cc5.getHantei5());
         // 点数を計算
            mochiten=mochiten+cc5.getReturnPoint()-kaketen;
            textField.setText(String.valueOf(mochiten));
         // ボタンを押せなくする
            btnNewButton.setEnabled(false); //koko
    }
}
```

図15．交換ボタンのイベント内容

④import文を追加しフィールド変数を変更します。//kokoの部分を追加します。

```
import jp.ict.aso.model.CardConfirmHtml5Bean;       //koko
import jp.ict.aso.model.CardHanteiHtml5Bean;        //koko
import jp.ict.aso.model.CardShuffleArrayListBean;   //koko
import jp.ict.aso.model.CardShuffleHtml10Bean;      //koko

public class Poker extends JFrame {
```

図16．import文の追加

```
private final Action action = new SwingAction();
private final Action action_1 = new SwingAction_1();
private final Action action_2 = new SwingAction_2();
private final Action action_3 = new SwingAction_3();

private ArrayList list;                             //koko
private CardShuffleArrayListBean csalb;             //koko
private CardShuffleHtml10Bean cs10;                 //koko
private CardHanteiHtml5Bean ch5;                    //koko

private JEditorPane editorPane;                     //koko
private JEditorPane editorPane_1;                   //koko
private JCheckBox chckbxNewCheckBox;                //koko
private JCheckBox chckbxNewCheckBox_1;              //koko
private JCheckBox chckbxNewCheckBox_2;              //koko
private JCheckBox chckbxNewCheckBox_3;              //koko
private JCheckBox chckbxNewCheckBox_4;              //koko
private JCheckBox chckbxNewCheckBox_5;              //koko
private JCheckBox chckbxNewCheckBox_6;              //koko
private JCheckBox chckbxNewCheckBox_7;              //koko
private JCheckBox chckbxNewCheckBox_8;              //koko
private JCheckBox chckbxNewCheckBox_9;              //koko
private String[] checkedCard = new String[5];       //koko
private JToggleButton tglbtnNewToggleButton;        //koko
private JToggleButton tglbtnNewToggleButton_1;      //koko
private JButton btnNewButton;                       //koko
private JButton btnNewButton_1;                     //koko

/**
 * Launch the application.
 */
```

図17．フィールド変数の追加

⑤ローカル変数の宣言になっている部分を変更します。**JEditorPane、JCheckBox、JToggleButton、JButtonのローカル宣言を削除**します。大量になるのでソースコードは省略します。

⑥cardShuffle()メソッドを定義します。

```
public void cardShuffle() {
    // カードをシャッフル
      csalb = new CardShuffleArrayListBean();
      csalb.shuffle();
      list=csalb.getList();
    // カード10枚をhtml化
      cs10 = new CardShuffleHtml10Bean();
      cs10.setList(list);
      editorPane.setText(cs10.getShuffleCardImage10());

    // 初期の５枚の役を判定
      ch5 = new CardHanteiHtml5Bean();
      ch5.setList(list);
      ch5.hanteiCard5();
      editorPane_1.setText(ch5.getHantei5());
}
```

図18．cardShuffle()メソッドの内容

⑦cardCheck()メソッドを追加します。

```
public void cardCheck() {  //チェックされたカードを配列に保存
    //最初にチェック済みカード配列の初期化
    for (int j = 0; j < checkedCard.length; j++) { //koko
        checkedCard[j] = "";                       //koko
    }
    int i=0;
    if(chckbxNewCheckBox.isSelected()) {
        checkedCard[i]= (list.get(0)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_1.isSelected()) {
        checkedCard[i]= (list.get(1)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_2.isSelected()) {
        checkedCard[i]= (list.get(2)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_3.isSelected()) {
        checkedCard[i]= (list.get(3)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_4.isSelected()) {
        checkedCard[i]= (list.get(4)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_5.isSelected()) {
        checkedCard[i]= (list.get(5)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_6.isSelected()) {
        checkedCard[i]= (list.get(6)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_7.isSelected()) {
        checkedCard[i]= (list.get(7)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_8.isSelected()) {
        checkedCard[i]= (list.get(8)).toString() ;  i++;
    }
    if(chckbxNewCheckBox_9.isSelected()) {
        checkedCard[i]= (list.get(9)).toString() ;  i++;
    }
}
```

図19．cardCheck()メソッドの内容

⑧作成済みのカードの画像を配置します。

図20．カード画像の配置

⑦実行確認します。エディタの画面内で右クリック → 実行 → Javaアプリケーションで実行されます。

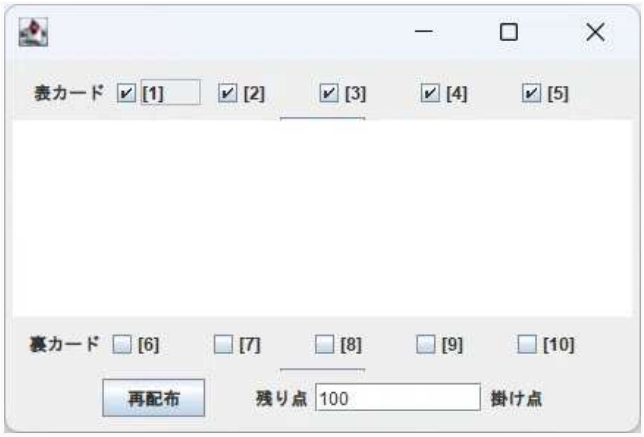

図21．ポーカー初期起動

⑧フレームを広げて再配布ボタンをクリックして動きを確認してください。

図22．ポーカーフレームの大きさの変更

図23．カード初期交換状態

図22、図23のようにとりあえず動くことだけ確認してください。

実装変更

この状態では、様々な問題を含んでいます（起動時初期配布のカードが表示されていない、交換すると再配布ボタンを押せなくなる等々）。全て解消して問題なく動くよう修正してください。

特に、交換ボタンをクリックしたときの**入力ミスに対する処理を追加**してください。
【入力ミスに対する処理】
・チェックされたカードが4枚以下の時は再配布とします。
・チェックされたカードが6枚以上の時は6枚目以降は無視します。
・残り点の領域は編集不可にします。
・掛け点に数字以外が入力されたときはコンソールにメッセージを出力するだけで交換処理は行いません。

これで、完成です。

図24．ポーカーゲーム完成

テスト

最後にすべての役が適正に判定されるかテストします。テスト用のアプリケーションを作成して、すべての役と配点の結果を確認してください。
実は**役の表示に若干の問題**があります。何が問題なのか見つけてください。

テストを行うクラスは**CardScoreBeanクラス**です。以下の**クラス図**を参考に、CardScoreBeanを実体化するmainメソッドを持った**PokerScoreTest.java**を作成（テスト用スタブを作成）し、すべての役を判定してバグの内容を調査し、問題を報告してください。

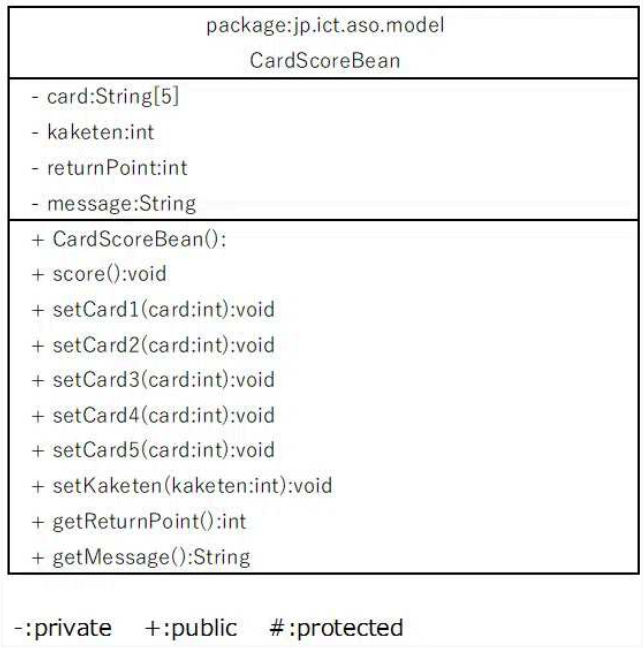

図25．CardScoreBeanクラス図

【ヒント】
図26のように作成するPokerScoreTest.javaは実行可能な（mainメソッドを含んだ）単純なクラスです。CardScoreBeanでは判定するカード５枚の情報をセッターメソッドの引数で与えます。引数で与えるカードの種類は以下のようになり、事前にソートしてセッターメソッドに設定します。
（カードの種類）
クラブ：101～113　ダイヤ：201～213　ハート：301～313
スペード：401～413

ソースコードの枠組みは以下のようになります。mainメソッド内でCardScoreBeanを実体化して、セッターメソッドに判定するカード番号の５つの組と掛け点を与えます。scoreメソッドで役と配点を計算させてゲッターメソッドで結果を取得し確認してください。

```
package jp.ict.aso.controller;
import jp.ict.aso.model.CardScoreBean;
public class PokerScoreTest {
    public static void main(String[] args) {
    }
}
```

図26．PokerScoreTestクラスの枠組み

単独起動

実行可能JARファイル

①せっかくですので、単独で起動できるアプリケーションにエクスポートしましょう。Java1.8以上のJREの環境がWindowsのPCにインストールされていればダブルクリックで起動できます。

Eclipseパッケージ・エクスプローラより
SwingBinaryDecimalプロジェクトを右クリック → エクスポート
→ Java → 実行可能JARファイル → 次へ
→ 以下のように設定する → 完了

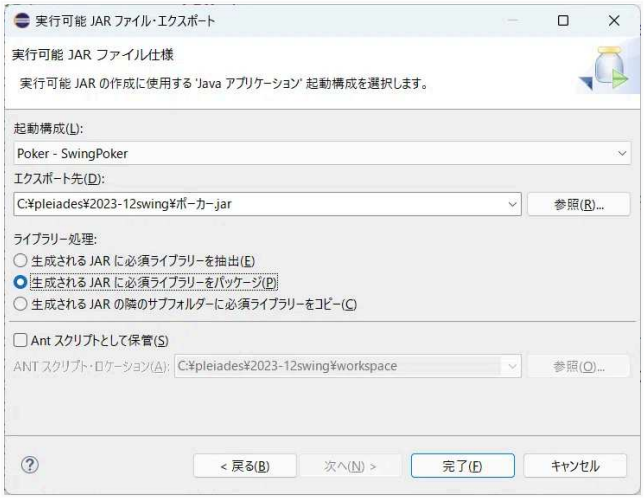

図27．実行可能JARファイルの設定

以下のような警告が出る場合がありますが、気にしません。

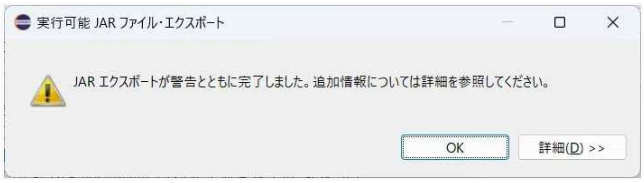

図28．警告ダイアログ

作成されたjarファイルをダブルクリックするとアプリが起動します。

図29．実行可能JARファイル

最後に

今回は、GUIデザイン部分の設定の詳細は提示していません。また、実装変更部分のソースコード例も載せていません。自由に工夫してみてください！