Übungsserie 6

Abgabe: Donnerstag 19.05.2022

Damit für alle die gleichen Regeln gelten, müssen Ihre Java-Programme mit dem OpenJDK 17 ohne Warnungen kompilieren. Vermeiden Sie insbesondere nicht autorisierte Packages oder Build-Environments. Reichen Sie Ihre Programme (Bitte nur `.java` *Dateien) als ZIP Archiv via Opal ein.*

Aufgabe 1 (2 Punkte) Mau Mau ist ein beliebtes Kartenspiel zum Zeitvertreib welches man sogar in Turnieren spielen kann. Das Spielprinzip ist einfach. Es gibt 32 Spielkarten mit den Farben *Kreuz*, *Pik*, *Herz* und *Karo* und Spielsymbolen: *7*, *8*, *9*, *10*, *Bube*, *Dame*, *König* und *Ass*. Zu Beginn des Spiels werden die Karten gemischt und jedem Spieler eine handvoll Karten ausgegeben. Eine Karte wird von den restlichen Karten abgehoben und aufgedeckt, die anderen Karten bilden einen Stapel, auch Talon genannt. Ein Spieler fängt an, und kann ggf. eine seiner Karten ablegen. Dies ist im einfachsten Spielmodi der Fall wenn sie das gleiche Symbol oder die gleiche Farbe hat. Falls keine Karte abgelegt werden kann, muss eine Strafkarte gezogen werden und der nächste Spieler ist dran. Der Spieler, welcher zu erst alle Karten ablegen konnte, hat gewonnen.[1]

Die gute Nachricht ist, folgender SourceCode steht bereit. Die weniger gute Nachricht ist, es werden darin u.U. Klassen genutzt welche Sie noch nicht kennen, ggf. die `LinkedList` Implementation einer doppelt verketteten Liste für einen Stack. Informieren Sie sich daher über alle Typen welche Sie noch nicht kennen.

(a) Lesen Sie sich in den Quellcode ein und versuchen Sie ihn nachzuvollziehen. Notieren Sie sich Fehler.

(b) Derzeit können Sie das Spiel nur gegen menschliche Nutzer spielen. Bilden Sie mittels Vererbung eine Klasse `ComputerPlayer` welche automatisch gegen Sie spielen kann.

(c) Nutzen Sie die existierende State-Machine `MauMau` um einen erweiterten Regelsatz `MauMauExt` zu implementieren.

- Bube auf Bube stinkt, darf also nicht gelegt werden
- Wird ein Bube gespielt so darf sich der auspielende Spieler einen Farbwert wünschen. Der Wunsch bleibt solange aufrecht erhalten bis eine Karte dieser Farbe gelegt wird.
- Bei 7 zwei ziehen, mit nochmaliger 7 verlängerbar (4 ziehen), usw.
- Bei Ass aussetzen oder mit weiterm Ass verlängern
- Achten Sie darauf das Ihre Computerspieler beide Spiele beherrschen.
- sonst bleiben die ursprünglichen Regeln erhalten.

Hinweis: Der Record `Move` ist schon so gebaut, dass das Wünschen einer Farbe möglich ist.

```java
import java.util.*;

enum Symbol {
  SEVEN("7"),
  EIGHT("8"),
  NINE ("9"),
  TEN  ("10"),
  JACK ("J"),
  QUEEN("Q"),
  KING ("K"),
  ACE  ("A");

  String symbol;

  Symbol(String symbol) {
    this.symbol = symbol;
  }

  @Override
  public String toString() {
    return symbol;
```

[1] Genau genommen muss dieser nach Abgabe der vorletzten Karte *Mau* gerufen haben und nach Abgabe der letzten Karte *MauMau*. Darauf wollen wir hier verzichten.

```
  }
}

enum Color {
  CLUB   ("♧"),
  SPADE  ("♤"),
  HEART  ("♡"),
  DIAMOND("♢");

  String color;

  Color(String color) {
    this.color = color;
  }

  @Override
  public String toString() {
    return color;
  }
}

enum Card {
  CLUB_SEVEN   (Color.CLUB   , Symbol.SEVEN),
  CLUB_EIGHT   (Color.CLUB   , Symbol.EIGHT),
  CLUB_NINE    (Color.CLUB   , Symbol.NINE ),
  CLUB_TEN     (Color.CLUB   , Symbol.TEN  ),
  CLUB_JACK    (Color.CLUB   , Symbol.JACK ),
  CLUB_QUEEN   (Color.CLUB   , Symbol.QUEEN),
  CLUB_KING    (Color.CLUB   , Symbol.KING ),
  CLUB_ACE     (Color.CLUB   , Symbol.ACE  ),
  SPADE_SEVEN  (Color.SPADE  , Symbol.SEVEN),
  SPADE_EIGHT  (Color.SPADE  , Symbol.EIGHT),
  SPADE_NINE   (Color.SPADE  , Symbol.NINE ),
  SPADE_TEN    (Color.SPADE  , Symbol.TEN  ),
  SPADE_JACK   (Color.SPADE  , Symbol.JACK ),
  SPADE_QUEEN  (Color.SPADE  , Symbol.QUEEN),
  SPADE_KING   (Color.SPADE  , Symbol.KING ),
  SPADE_ACE    (Color.SPADE  , Symbol.ACE  ),
  HEART_SEVEN  (Color.HEART  , Symbol.SEVEN),
  HEART_EIGHT  (Color.HEART  , Symbol.EIGHT),
  HEART_NINE   (Color.HEART  , Symbol.NINE ),
  HEART_TEN    (Color.HEART  , Symbol.TEN  ),
  HEART_JACK   (Color.HEART  , Symbol.JACK ),
  HEART_QUEEN  (Color.HEART  , Symbol.QUEEN),
  HEART_KING   (Color.HEART  , Symbol.KING ),
  HEART_ACE    (Color.HEART  , Symbol.ACE  ),
  DIAMOND_SEVEN(Color.DIAMOND, Symbol.SEVEN),
  DIAMOND_EIGHT(Color.DIAMOND, Symbol.EIGHT),
  DIAMOND_NINE (Color.DIAMOND, Symbol.NINE ),
  DIAMOND_TEN  (Color.DIAMOND, Symbol.TEN  ),
  DIAMOND_JACK (Color.DIAMOND, Symbol.JACK ),
  DIAMOND_QUEEN(Color.DIAMOND, Symbol.QUEEN),
  DIAMOND_KING (Color.DIAMOND, Symbol.KING ),
  DIAMOND_ACE  (Color.DIAMOND, Symbol.ACE  );

  final Color  color;
  final Symbol symbol;

  Card(Color color, Symbol symbol) {
    this.color = color;
    this.symbol = symbol;
```

```
  }

  @Override
  public String toString() {
    return color + "" + symbol;
  }
}

record Move(Card card, Color color) {}

interface MauMauable {
  void putOnTop(Card c);
  public int getPenalty();
  boolean isAllowed(Card c);
  void apply(Move m);
  Card top();
}

class MauMau implements MauMauable {
  Card top;

  @Override
  public void putOnTop(Card c) {
    top = c;
  }

  @Override
  public int getPenalty() {
    return 1;
  }

  @Override
  public boolean isAllowed(Card c) {
    if(top.symbol == c.symbol || top.color == c.color) {
       return true;
    }
    return false;
  }

  @Override
  public void apply(Move m) {
    if( !isAllowed(m.card()) ) {
      throw new RuntimeException("Player unallowed card, nothing is done.");
    }
    putOnTop(m.card());
    System.out.println("played: " + m.card());
  }

  @Override
  public Card top() {
    return top;
  }

  @Override
  public String toString() {
    StringBuilder statusLine = new StringBuilder();
    statusLine.append("# top-card: " + top);
    return statusLine.toString();
  }
}
```

```
class Game {
  MauMauable state;
  LinkedList<Card> talon;
  Player[] players;
  Player winner;

  Game(Player[] players, MauMauable state) {
    this.players = players;
    talon = new LinkedList<Card>(Arrays.asList(Card.values()));
    Collections.shuffle(talon);

    int cards_per_player = 5;
    this.state = state;
    state.putOnTop(talon.pop());
    for( Player p : players ) {
      p.setHand(talon.subList(0, 5).toArray(new Card[0]));
      talon.subList(0, 5).clear();
    }
    winner = null;
  }

  void start() {
    System.out.println("New game starts:");
    while( winner == null ) {
      for( Player p : players ) {
        System.out.println(p.getName() + " make your choice: ");
        Move m = p.ask(state);
        if( m.card() == null ) {
          int draws = state.getPenalty();
          System.out.println(p.getName() + " draws " + draws);
          for(int i = 0; i < draws; ++i) {
            if( !talon.isEmpty() ) {
              p.add(talon.pop());
            }
          }
          continue;
        }
        if( p.remove(m.card()) == 0 ) {
          winner = p;
          return;
        }
        talon.push(state.top());
        Collections.shuffle(talon);
        state.apply(m);
      }
    }
  }
}

class Player {
  String name;
  LinkedList<Card> hand;
  Player(String name) {
    this.name = name;
  }

  String getName() {
    return name;
  }

  void setHand(Card... crds) {
```

```
    hand = new LinkedList<Card>(Arrays.asList(crds));
    Collections.sort(hand);
  }

  public String toString() {
    return name + ": " + hand;
  }

  Move ask(MauMauable state) {
    int card_num;
    do {
      System.out.println(state);
      System.out.println("you have the cards: " + hand);
      System.out.println("make your choice: 0-" + (hand.size()-1) + " (-1 for draw):");
      Scanner in = new Scanner(System.in);
      card_num = in.nextInt();
      if( card_num <= -1 || card_num >= hand.size() ) {
        return new Move(null, null);
      }
    } while( !state.isAllowed(hand.get(card_num)) );
    Card c = hand.get(card_num);
    return new Move(c, c.symbol == Symbol.JACK ? chooseColor() : null);
  }

  int remove(Card c) {
    hand.remove(c);
    return hand.size();
  }

  void add(Card c) {
    hand.push(c);
    Collections.sort(hand);
  }

  Color chooseColor() {
    int color_num;
    do {
      System.out.println("choose a color: 0.." + (Color.values().length - 1) + " -> ♤,
↪  ♤, ♡, ♢ : ");
      Scanner in = new Scanner(System.in);
      color_num = in.nextInt();
    } while( color_num < 0 || color_num >= Color.values().length );
    return Color.values()[color_num];
  }
}

public class cards {
  public static void main(String[] args) {
    Game g = new Game(new Player[] {
      new Player("Human1"),
      new ComputerPlayer("Computer1")},
      new MauMauExt());
    g.start();
    System.out.println(g.winner.getName() + " has won.");
  }
}
```