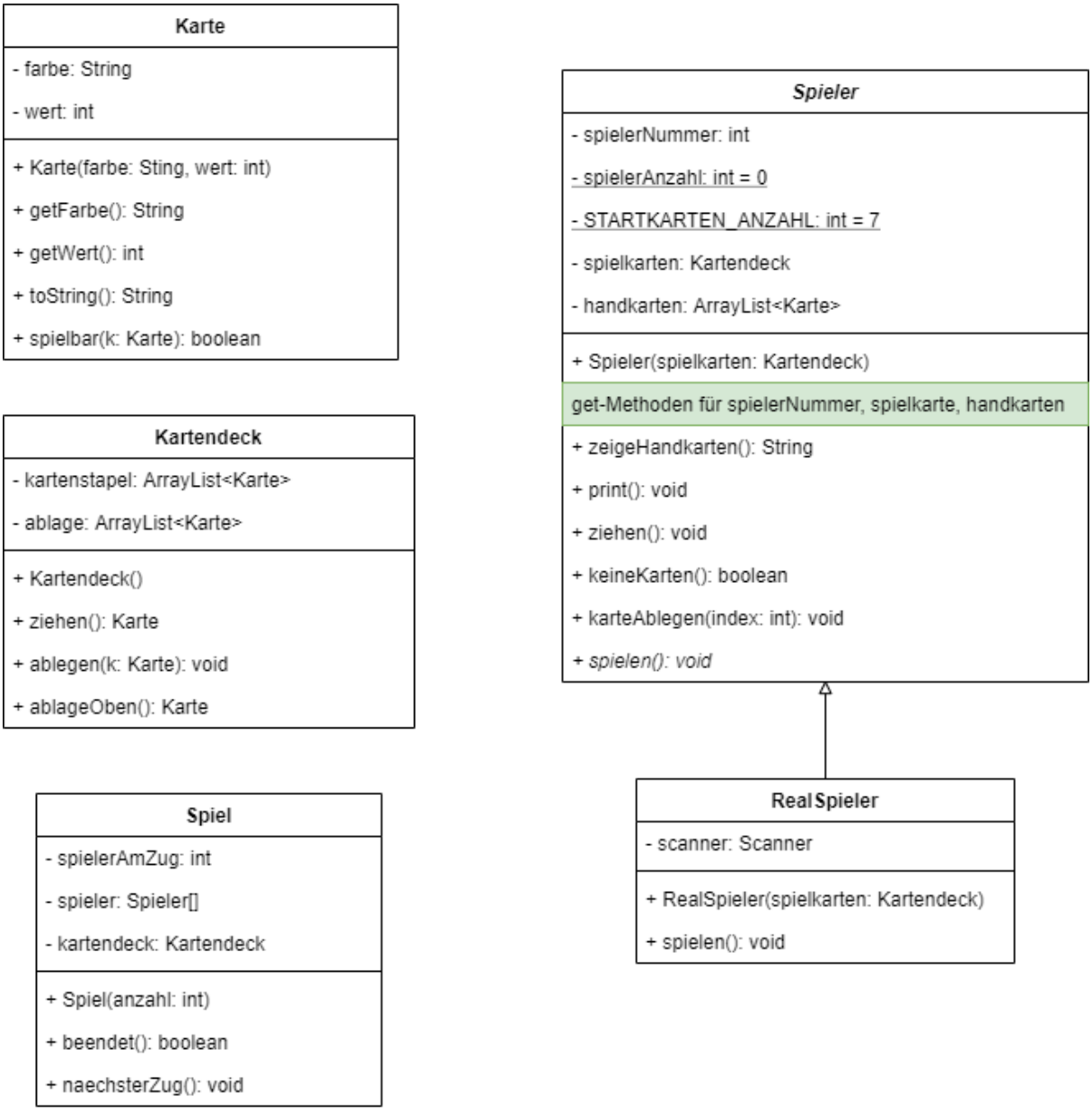
Übungsprüfung

Aufgabe 1 [80 Punkte]

Setzen Sie ein vereinfachtes Mau-Mau Spiel um. Gespielt wird mit Spielkarten mit den üblichen Farben „Herz“, „Karo“, „Kreuz“, „Pik“ und Werten von 7 bis 14 (die Werte 11-14 stehen für Bube, Dame, König, Ass). Die Spielkarten befinden sich gemischt in einem Deck und eine Karte wird zu Beginn aufgedeckt. Die Spieler ziehen eine Anzahl von Startkarten. Die Spieler sind reihum an der Reihe. Ein zufälliger Spieler beginnt. Ist der Spieler an der Reihe, kann er/sie entweder eine Karte ablegen, oder eine Karte ziehen. Karten können abgelegt werden, wenn sie entweder in ihrer Farbe, oder ihrem Wert mit der obersten Karte der Ablage übereinstimmen. Wird versucht eine Karte abzulegen, die dieser Regel nicht folgt, ist das Ablegen nicht erfolgreich und der Spieler zieht stattdessen eine Karte. Ist der Kartenstapel leer, werden die Ablagekarten bis auf die oberste gemischt dem Kartenstapel hinzugefügt. Hat einer der Spieler keine Karten mehr, ist das Spiel beendet.



Aufgabe 1.1 [10 Punkte]

Realisieren Sie die Klasse Karte:

- Legen Sie die Attribute an [2 Punkte]
- Legen Sie den Konstruktor an, in dem die Attribute gesetzt werden [2 Punkte]
- Legen Sie get-Methoden für die Attribute an [2 Punkte]
- toString-Methode: gibt einen String zurück der Farbe und Wert mit Leerzeichen getrennt beinhaltet [1 Punkt]
- spielbar-Methode: Gibt zurück, ob die im Parameter übergebene Karte auf die Karte welche die Methode aufruft gespielt werden kann -> Es müssen dazu Farbe oder Wert übereinstimmen [3 Punkte]

Code für main-Methode zum Testen:

```
Karte karte = new Karte("Karo", 8);
System.out.println(karte.getFarbe());
System.out.println(karte.getWert());
System.out.println(karte);
System.out.println(karte.spielbar(new Karte("Karo", 10)));
System.out.println(karte.spielbar(new Karte("Kreuz", 8)));
System.out.println(karte.spielbar(new Karte("Kreuz", 10)));
```

Erwartete Ausgabe:

```
Karo
8
Karo 8
true
true
false
```

Aufgabe 1.2 [20 Punkte]

Realisieren Sie die Klasse Kartendeck:

- Klassengerüst mit Attributen und Methoden umsetzen [3 Punkte]
- Konstruktor:
  - Attribute initialisieren als leere Listen [2 Punkt]
  - Alle Spielkarten erzeugen (eine für jede Farbe „Herz“, „Karo“, „Kreuz“, „Pik“ kombiniert mit jedem Wert von 7 bis 14) [3 Punkte] und dem Kartenstapel hinzufügen [1 Punkt] in zufälliger Reihenfolge [2 Punkte]
  - Eine Karte ziehen und sie direkt ablegen (siehe weitere Methoden) [1 Punkt]
- ziehen-Methode:
  - Die vorderste Karte vom Kartenstapel entfernen und zurückgeben [2 Punkte]
  - Wenn der Kartenstapel leer ist zuvor alle Karten der Ablage, bis auf die oberste, entfernen und dem Kartenstapel hinzufügen [2 Punkte] in zufälliger Reihenfolge [1 Punkt]
- ablegen-Methode: Die übergebene Karte dem Ablagestapel hinzufügen, so dass Sie obenauf liegt [2 Punkte]
- ablageOben-Methode: Die oberste Karte des Ablagestapels zurückgeben (ohne Sie dabei zu entfernen) [2 Punkte]

Code für main-Methode zum Testen:

```
Kartendeck kartendeck = new Kartendeck();
Karte k = kartendeck.ziehen();
System.out.println(k);
kartendeck.ablegen(k);
System.out.println(kartendeck.ablageOben());
```

Erwartete Ausgabe: Zufällige Karte, aber beide Male die Gleiche
```
Karo 11
Karo 11
```

Aufgabe 1.3 [27 Punkte]

Realisieren Sie die abstrakte Klasse Spieler:

- Klassengerüst mit Attributen und Methoden umsetzen [3 Punkte]
- Konstruktor: Initialisieren Sie die Attribute
  - Speichern Sie die übergebene Referenz aufs Kartendeck [1 Punkt]
  - Weißen Sie dem Spieler eine Nummer zu (erster Spieler 1, zweiter Spieler 2, usw.); Speichern Sie dazu die Anzahl der Spieler [2 Punkte]
  - Legen Sie eine Liste für die Handkarten an und fügen Sie dieser durch ziehen die geforderte Anzahl an Startkarten hinzu (siehe weitere Methoden) [3 Punkt]
- Legen Sie get-Methoden für spielerNummer, spielkarten und handkarten an [2 Punkt]
- zeigeHandkarten-Methode: Die Methoden gibt einen über StringBuilder erzeugten String zurück in dem alle Handkarten durch | getrennt aufgelistet werden mit ihrer Position in der Liste davor, z. B. | 0: Herz 8 | 1: Karo 7 | [3 Punkt]
- print-Methode: Gibt auf der Konsole die Handkarten aus und die oberste Karte der Ablage. Beispielausgabe: [2 Punkt]
  ```
  Handkarten: | 0: Herz 8 | 1: Karo 7 |
  Ablage: Pik 10
  ```
- ziehen-Methode: Zieht vom Kartendeck eine Karte und fügt sie den Handkarten hinzu [2 Punkt]
- keineKarten-Methode: Gibt zurück, ob der Spieler keine Handkarten mehr besitzt [1 Punkt]
- karteAblegen-Methode: Wenn die Karte am angegebenen Index in den Handkarten existiert und auf die oberste Karte der Ablage gespielt werden darf, wird sie abgelegt. Ansonsten zieht der Spieler eine Karte. [6 Punkt]
  Das Geschehen wird über eine Ausgabe auf der Konsole dokumentiert, z. B.
  ```
  Spieler 1 legt eine Karte
  ```
  oder
  ```
  Sie können diese Karte nicht spielen
  ```
- Legen Sie die abstrakte spielen-Methode an [2 Punkt]

Aufgabe 1.4 [8 Punkte]

Realisieren Sie die Klasse RealSpieler, die von Spieler erbt:

- Klassengerüst mit Attributen und Methoden umsetzen [2 Punkt]
- Konstruktor: Initialisiert die Attribute [2 Punkt]
- spielen-Methode: [4 Punkte]
  - Geben Sie auf der Konsole aus:
    ```
    Wollen Sie eine Karte spielen? j: ja, n: nein
    ```
  - Lesen Sie die Antwort von der Konsole ein
  - Wenn keine Karte gespielt werden soll, zieht der Spieler eine Karte, ansonsten wird über die Konsole gefragt, welchen Index die Karte hat die gespielt werden soll, um diese dann entsprechend der Eingabe abzulegen:
    ```
    Nummer der Karte?
    ```

Code für main-Methode zum Testen von Spieler und RealSpieler:
```
RealSpieler spieler = new RealSpieler(new Kartendeck());
System.out.println(spieler.getSpielerNummer());
Kartendeck deck = spieler.getSpielkarten();
System.out.println(spieler.getHandkarten());
```

```
System.out.println(spieler.zeigeHandkarten());
spieler.print();
spieler.ziehen();
System.out.println(spieler.zeigeHandkarten());
System.out.println(spieler.keineKarten());
spieler.karteAblegen(0);
System.out.println(spieler.zeigeHandkarten());
spieler.spielen();
```

Erwartete Ausgabe: (Beispieldaten, karteAblegen nicht erfolgreiche, spielen erfolgreich)

```
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
1
[Karo 11, Kreuz 11, Pik 14, Karo 7, Herz 14, Kreuz 7, Herz 12]
| 0: Karo 11 | 1: Kreuz 11 | 2: Pik 14 | 3: Karo 7 | 4: Herz 14 | 5: Kreuz
7 | 6: Herz 12 |
Handkarten: | 0: Karo 11 | 1: Kreuz 11 | 2: Pik 14 | 3: Karo 7 | 4: Herz 14
| 5: Kreuz 7 | 6: Herz 12 |
Ablage: Pik 10
Spieler 1 zieht eine Karte.
| 0: Karo 11 | 1: Kreuz 11 | 2: Pik 14 | 3: Karo 7 | 4: Herz 14 | 5: Kreuz
7 | 6: Herz 12 | 7: Kreuz 14 |
false
Sie können diese Karte nicht spielen.
Spieler 1 zieht eine Karte.
| 0: Karo 11 | 1: Kreuz 11 | 2: Pik 14 | 3: Karo 7 | 4: Herz 14 | 5: Kreuz
7 | 6: Herz 12 | 7: Kreuz 14 | 8: Kreuz 13 |
Wollen Sie eine Karte spielen? j: ja, n: nein
j
Nummer der Karte?
2
Spieler 1 legt eine Karte.
```

Aufgabe 1.5 [10 Punkte]

Realisieren Sie die Klasse Spiel:

- Klassengerüst mit Attributen und Methoden umsetzen [1 Punkt]
- Konstruktor:
  - Kartendeck anlegen und Spieler, entsprechend der im Parameter übergebenen Anzahl an Spielern [3 Punkte]
  - Spieler der am Zug ist zufällig aus allen Spielern auswählen und dessen Position im Spielerarray merken [1 Punkt]
  - beendet-Methode: Gibt zurück, ob einer der Spieler keine Handkarten mehr hat und das Spiel damit beendet ist [2 Punkte]
  - naechsterZug-Methode: Gibt aus, welcher Spieler am Zug ist, z. B.
    `Spieler 1 ist am Zug`
    Ruft die print-Methode und danach die spielen-Methode des Spielers auf [2 Punkte] und ermittelt dann den nächsten Spieler, der am Zug ist. [1 Punkt]

Code für main-Methode zum Testen des Spiels:

```
Spiel spiel = new Spiel(2);
while(!spiel.beendet()){
    spiel.naechsterZug();
}
System.out.println("Spiel beendet.");
```

Erwartete Ausgabe: (Beispieldaten)

```
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 1 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 2 zieht eine Karte.
Spieler 1 ist am Zug.
Handkarten: | 0: Pik 7 | 1: Karo 14 | 2: Pik 9 | 3: Pik 14 | 4: Karo 11 |
5: Pik 11 | 6: Karo 12 |
Ablage: Pik 10
Wollen Sie eine Karte spielen? j: ja, n: nein
j
Nummer der Karte?
3
Spieler 1 legt eine Karte.
Spieler 2 ist am Zug.
...
```

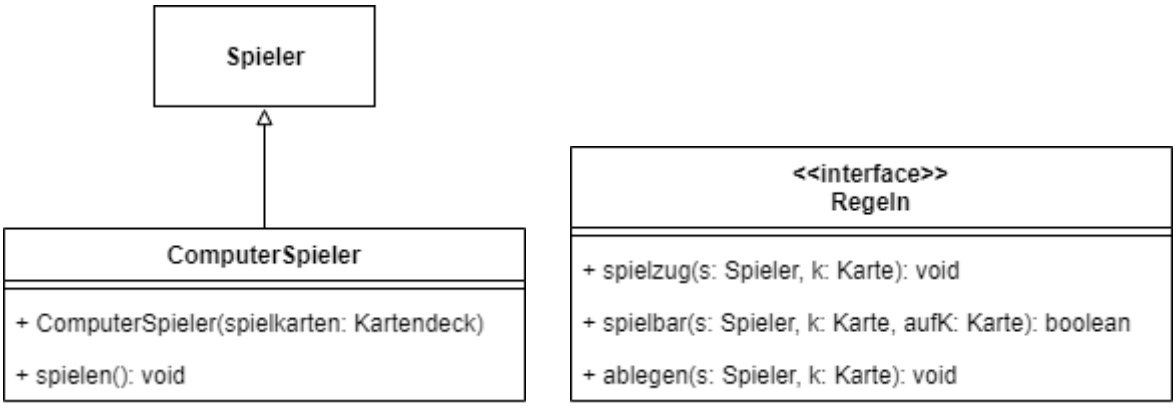
Code Qualität [5 Punkte]

Beachten Sie beim Schreiben des Codes:

- den Style Guide [2 Punkte]
- leserlichen und möglichst effizienten Code zu schreiben (ggf. Kommentare) [2 Punkte]

Aufgabe 2 [20 Punkte]

Erweitern Sie Aufgabe 1 um einen Computerspieler und um ein Regelwerk.



Aufgabe 2.1 [7 Punkte]

Realisieren Sie die Klasse ComputerSpieler. Der Computer-Spieler legt die erste passende Karte ab, die in den Handkarten zu finden ist und zieht eine Karte falls keine Handkarte passt. [5 Punkte]
Passen Sie die Spiel Klasse so an, dass im Konstruktor zwei Anzahl Werte übergeben werden, eine für die anzahl an Real-Spielern und eine für die Anzahl an Computer-Spielern. Legen Sie die Spieler entsprechend an. [2 Punkte]

Aufgabe 2.2 [3 Punkte]

Realisieren Sie das Interface Regeln entsprechend des Klassendiagramms.

Aufgabe 2.3 [5 Punkte]

Legen Sie eine Klasse „RegelnEinfach“ an, welches das Interface implementiert. Dieses soll die einfachen Regeln des Spiels umsetzen, wie sie oben beschrieben wurden (solange eine Karte in Wert oder Farbe mit der oberen Karte der Ablage übereinstimmt, kann sie abgelegt werden).

- spielzug-Methode: Ruft die spielen Methode des Spielers auf
- spielbar-Methode: Prüft, ob die Karte k auf die Ablage-Karte aufK abgelegt werden kann
- ablegen-Methode: Hat beim einfachen Regelwerk keine Funktion

Aufgabe 2.4 [5 Punkte]

Passen Sie Ihren bestehenden Code an, um das Regelwerk einzubinden:

- Erweitern Sie den Konstruktor von Spiel und Spieler um einen „Regel regel“ Parameter, welcher in beiden Klassen jeweils in einem passenden Attribut gespeichert werden soll.
- Legen Sie in der main-Methode ein RegelEinfach Objekt an und übergeben Sie dieses beim Anlegen des Spiel Objekts, welches es an die Spieler durchreicht.
- Rufen Sie in der Spiel-Klasse in der naechsterZug-Methode die spielzug Methode des regel Attributs auf, anstatt der spielen Methode des Spielers.
- Legen Sie in der Spieler-Klasse eine get-Methode für das regeln Attribut an. Ändern Sie zudem die karteAblegen-Methode in der Spieler-Klasse so, dass die Spielbarkeit einer Karte über die entsprechende Regel Methode geprüft wird und dass beim Ablegen einer Karte zusätzlich die ablegen-Methode der Regel aufgerufen wird (um je nach Regelwerk auf die abgelegte Karte reagieren zu können).

Code für main-Methode zum Testen der Aufgabe 2:

```
Regeln regeln = new RegelnEinfach();
Spiel spiel = new Spiel(1, 1, regeln);
while(!spiel.beendet()){
    spiel.naechsterZug();
}
System.out.println("Spiel beendet.");
```

Zusatzpunkte

Sie können weitere Zusatzpunkte erlangen, mit denen Sie eventuelle Fehler aus Aufgabe 1 und Aufgabe 2 ausgleichen können. Erweitern Sie dazu Ihren Code um folgende Funktionen:

- Implementieren Sie eine weitere Klasse „RegelnStandard“, welche das Regeln-Interface implementiert und folgende Regeln umsetzt:
  - Wird eine 7 abgelegt, muss der nächste Spieler 2 Karten ziehen.
  - Wird eine 8 abgelegt, muss der nächste Spieler aussetzen.
  - Wird ein Bube (11) abgelegt, darf der Spieler sich eine Farbe wünschen, die bedient werden muss (nächste abgelegte Karte muss von dieser Farbe sein).
  - Ein Bube darf immer gespielt werden, nur nicht auf einen anderen Buben.

  Ergänzen Sie für das Wünschen einer Farbe eine entsprechende abstrakte Methode in Spieler, welche in RealSpieler und ComputerSpieler unterschiedlich umgesetzt wird (der Real-Spieler wird nach seinem Wunsch gefragt, der Computer-Spieler nimmt die Farbe der ersten Handkarte die kein Bube ist).
- Optimieren Sie den Computer-Spieler, so dass dieser strategischer spielt. Dabei können zum Beispiel Querverbindungen zwischen Karten berücksichtigt werden, um bei Karten mit gleichem Wert von einer Farbe zu einer anderen wechseln zu können. Es kann außerdem das Regelwerk berücksichtigt werden, indem zum Beispiel bei den Standard-Regeln ein Bube zur passenden Zeit gelegt wird und sich die Farbe gewünscht wird, die am häufigsten in den Handkarten vorkommt.