Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, N. Lommen, D. Meier, F. Meyer

Aufgabe 3 (Einfache Klassen): (2 + 2 + 4 + 3 + 4 + 3 + 2 + 6 + 2 + 2 = 30 Punkte)

In dieser Aufgabe geht es um das bekannte Kartenspiel Mau-Mau. Dieses Spiel wird mit einem Skatblatt aus 32 Karten gespielt: Es gibt acht verschiedene Werte (Sieben, Acht, Neun, Zehn, Bube, Dame, König und Ass) in vier verschiedenen sog. Farben (Kreuz, Pik, Herz und Karo). Jede Kombination aus Wert und Farbe kommt in einem Skatblatt genau einmal vor. Wir konzentrieren uns hier hauptsächlich auf den Aspekt des *Bedienens*, d.h. der Festlegung, welche Karten aufeinander gespielt werden dürfen: Eine Karte k' darf genau dann auf eine Karte k gespielt werden (auch: k' bedient k), wenn mindestens eine der folgenden Bedingungen erfüllt ist:

- Der Wert von k stimmt mit dem Wert von k' überein.
- Die Farbe von k stimmt mit der Farbe von k' überein.
- Der Wert von k' ist Bube.

Beispiel: Das Herz-Ass bedient u.a. die Herz-Neun und das Kreuz-Ass, nicht aber den Karo-König. Der Pik-Bube bedient jede Karte, aber nicht jede Karte bedient den Pik-Buben.
Die weiteren Regeln von Mau-Mau sind in dieser Aufgabe nicht von Bedeutung.

Hinweise:

- Sie dürfen in allen Teilaufgaben beliebige Hilfsmethoden schreiben.
- Um sich zunächst auf die Erstellung der Klassen zu konzentrieren, müssen Sie in dieser Aufgabe die Prinzipien der Datenkapselung noch nicht beachten.

a) Schreiben Sie jeweils einen Aufzählungstyp (d.h. eine `enum`-Klasse) `Farbe` und `Wert` für die vier verschiedenen Farben bzw. die acht verschiedenen Werte. Verwenden Sie dabei für die Bezeichner der einzelnen Objekte ausschließlich Großbuchstaben. Umgehen Sie Umlaute, indem sie diese wie in Kreuzworträtseln üblich durch zwei Buchstaben kodieren.

b) Schreiben Sie eine `record`-Klasse `Karte` mit je einem Attribut vom Typ `Farbe` und `Wert`. Überschreiben Sie außerdem in dieser Klasse die Methode `String toString()`, die für jedes Objekt vom Typ `Karte` einen `String` ausgibt: Dazu sollen die Farbe und der Wert in dieser Reihenfolge konkateniert werden. So soll bspw. für das Herz-Ass (mit dem `Farbe`-Attribut `Farbe.HERZ` und dem `Wert`-Attribut `Wert.ASS`) der String `HERZASS` ausgegeben werden. Sie können hier und in allen folgenden Teilaufgaben davon ausgehen, dass die Attribute eines Objekts vom Typ `Karte` stets sinnvoll gesetzt sind, wenn auf diesem eine Methode aufgerufen wird oder es als Parameter übergeben wird.

Hinweise:

- Um die String-Repräsentation eines `enum`-Objekts zu erhalten, können Sie die bereits vorhandene Methode `toString()` auf Objekten eines `enum`-Typs nutzen. Bspw. ist `Farbe.HERZ.toString()` der String `"HERZ"`.

c) Ergänzen Sie die Klasse `Karte` um eine statische Methode `Karte neueKarte(Farbe f, Wert w)`, die ein neues Objekt vom Typ `Karte` mit den übergebenen Attributen erzeugt und zurückgibt. Nutzen Sie diese Methode, um eine weitere statische Methode `Karte neueKarte(String f, String w)` in der Klasse `Karte` zu schreiben. Diese soll ebenfalls ein neues Objekt vom Typ `Karte` zurückgeben, wobei diesmal je ein `String` für die zu setzende Farbe und den zu setzenden Wert übergeben werden. Sie können davon ausgehen, dass nur solche `String`s übergeben werden, für die auch ein zugehöriges `enum`-Objekt existiert.

d) Ergänzen Sie die Klasse `Karte` um die statische Methode `int kombinationen()`, die die Anzahl der verschiedenen Farbe-Wert-Kombinationen zurückgibt. Gestalten Sie Ihre Implementierung so, dass die Ausgabe auch dann noch korrekt ist, wenn sich die zugrundeliegenden `enum`-Klassen ändern. Nutzen Sie die Methode `kombinationen()`, um eine weitere statische Methode `Karte[] skatblatt()` in der Klasse `Karte` zu schreiben. Diese soll ein Array mit Elementen vom Typ `Karte` zurückgeben, in dem sich für jede Farbe-Wert-Kombination genau eine entsprechende Karte befindet. Das Array soll keine weiteren Elemente haben, insbesondere keine `null`-Elemente.

e) Ergänzen Sie die Klasse `Karte` um eine Methode `boolean bedient(Karte other)`. Beim Aufruf dieser Methode auf einer Karte `this` soll genau dann `true` zurückgegeben werden, wenn die Karte `this` die Karte `other` bedient. Wann eine Karte eine andere bedient, haben wir zu Beginn dieser Aufgabe definiert.
Angenommen, das Objekt `k1` enthält `Farbe.HERZ` im Attribut `farbe` und `Wert.BUBE` im Attribut `wert`. Außerdem enthalte das Objekt `k2` `Farbe.KARO` im Attribut `farbe` und `Wert.KOENIG` im Attribut `wert`. Der Aufruf `k1.bedient(k2)` soll dann `true` zurückgeben und der Aufruf `k2.bedient(k1)` soll `false` zurückgeben.
Nutzen Sie die Methode `bedient`, um eine weitere Methode `boolean bedienbar(Karte... kn)` in der Klasse `Karte` zu schreiben. Diese soll genau dann `true` zurückgeben, wenn mindestens eines der übergebenen `Karte`-Objekte in `kn` dasjenige `Karte`-Objekt bedient, auf dem die Methode aufgerufen wurde. Sie können hierbei davon ausgehen, dass `kn` nicht `null` ist.

f) Ergänzen Sie die Klasse `Karte` um eine statische Methode `void druckeDoppelBedienungen()`. Diese Methode soll alle Paare von `Karte`-Objekten mit unterschiedlichen Attributen durchgehen und für jedes Paar (`k1`,`k2`) eine Meldung ausgeben, wenn sowohl `k1.bedient(k2)` als auch `k2.bedient(k1)` gilt. Die Meldung soll mit `System.out.println` ausgegeben werden und folgende Form haben: `KARODAME bedient KAROKOENIG und KAROKOENIG bedient KARODAME.`

Hinweise:

- Um zu überprüfen, ob die Attribute von zwei Karten übereinstimmen, verwenden Sie die Methode `boolean equals(Karte k)`. Für zwei Karten `k1` und `k2` können Sie beispielsweise durch den Aufruf `k1.equals(k2)` überprüfen, ob alle Attribute übereinstimmen. Da Sie eine `record`-Klasse verwenden, steht die `equals`-Methode bereits automatisch zur Verfügung.

g) Schreiben Sie eine Klasse `Spieler` mit einem Attribut `kartenhand`, das ein Array mit Elementen vom Typ `Karte` ist. Außerdem soll ein zweites Attribut den Namen des Spielers enthalten und ein drittes Attribut `gespielteKarte` die Karte beinhalten, welche der Spieler zuletzt gespielt hat. Wählen Sie für diese beiden Attribute sinnvolle Typen. Schreiben Sie außerdem die Methode `String toString()` in der Klasse `Spieler`, die für jedes Objekt vom Typ `Spieler` den Namen des Spielers als `String` ausgibt. Sie können hier und in allen folgenden Teilaufgaben davon ausgehen, dass die Attribute eines Objekts vom Typ `Spieler` stets sinnvoll gesetzt sind, wenn auf diesem eine Methode aufgerufen wird oder es als Parameter übergeben wird.

h) Ergänzen Sie die Klasse `Spieler` außerdem um eine Methode `void spieleKarte(Karte k)`. Beim Aufruf dieser Methode soll geprüft werden, ob der Spieler mit seinen Karten die Karte `k` bedienen kann. Ist dies der Fall, dann soll sein Attribut `gespielteKarte` auf die erste Karte des Arrays `kartenhand` gesetzt werden, die `k` bedient, und die gespielte Karte soll aus seiner Hand entfernt werden. Geben Sie außerdem mit `System.out.println` eine geeignete Ausgabe aus (z.B.: `Max bedient KAROKOENIG mit KARODAME`). Kann der Spieler nicht bedienen, setzen Sie das Attribut `gespielteKarte` auf `null`. Geben Sie in diesem Fall keine Ausgabe aus.

Hinweise:

- Um die Karte aus der Hand zu entfernen, ist es sinnvoll, zuerst die Position zu bestimmen, an der sich die Karte befindet (z.B. mit einer `while`-Schleife). In einer zweiten Schleife wird nun ein neues Array mit allen Karten außer eben der zuvor bestimmten Karte gefüllt.

i) Ergänzen Sie die Klasse `Spieler` um eine `main`-Methode mit der bekannten Signatur. Erstellen Sie in dieser zuerst zwei `Spieler`-Objekte für die Spieler Elisabeth und Klaus. Elisabeth hat die Karten Herz-Neun, Herz-Zehn und Pik-Bube in ihrer Hand. Klaus hingegen hat die Karten Herz-Zehn, Pik-Bube und Herz-Neun. (Beachten Sie die Reihenfolge!) Erstellen Sie hierzu jeweils ein dreielementiges Karten-Array für die `kartenhand`. Lassen Sie das Attribut `gespielteKarte` uninitialisiert.

j) Erweitern Sie die Klasse `Spieler` um eine Methode `void spiele(Karte k)`. Diese Methode simuliert ein vereinfachtes Mau-Mau-Spiel. Begonnen wird mit der Karte `k`. Eine Partie hat hierbei den folgenden Ablauf. Der Spieler versucht, die Karte `k` zu bedienen. Hierzu wird die Methode `void spieleKarte(Karte k)` verwendet. Kann der Spieler bedienen, so wird die Karte, mit welcher er bedienen konnte, aus

seiner Hand entfernt. Anschließend versucht der Spieler, nun die gerade entfernte Karte zu bedienen. Dies wird solange wiederholt, bis der Spieler entweder nicht mehr bedienen kann oder seine Kartenhand leer ist. Kann der Spieler nicht mehr bedienen, so wird `Elisabeth hat verloren!` ausgegeben für einen Spieler, der Elisabeth heißt. Hingegen wird `Elisabeth hat gewonnen!` ausgegeben für einen Spieler, der Elisabeth heißt, falls die Kartenhand zum Schluss leer ist. Sollte beim Aufruf der Methode `spiele` die Kartenhand bereits leer sein, so kann sich Ihre Methode beliebig verhalten. In der `main`-Methode soll zuerst Elisabeth und dann Klaus beginnend mit der Karo-Zehn spielen.

Hinweise:

- Die erzeugte Ausgabe sieht wie folgt aus:

```
Elisabeth bedient KAROZEHN mit HERZZEHN
Elisabeth bedient HERZZEHN mit HERZNEUN
Elisabeth bedient HERZNEUN mit PIKBUBE
Elisabeth hat gewonnen!
Klaus bedient KAROZEHN mit HERZZEHN
Klaus bedient HERZZEHN mit PIKBUBE
Klaus hat verloren!
```

Aufgabe 5 (Programmierung mit Datenabstraktion): (1 + 3 + 2 + 7 + 3 + 4 = 20 Punkte)

In dieser Aufgabe soll eine Java-Klasse erstellt werden, mit der sich Rechtecke repräsentieren lassen. Ein solches Rechteck lässt sich mit den Koordinaten x und y für die linke obere Ecke, der Breite *width* und der Höhe *height* beschreiben, wobei x, y, *width* und *height* ganze Zahlen sind. Die Breite und die Höhe eines Rechtecks können nicht negativ sein.
Ihre Implementierung sollte mindestens die folgenden Methoden beinhalten. Sie sollten dabei die *Prinzipien der Datenkapselung* berücksichtigen. Hilfsmethoden müssen als `private` deklariert werden. In dieser Aufgabe dürfen Sie die in der Klasse `Utils` zur Verfügung gestellten Hilfsfunktionen, aber keine Bibliotheksfunktionen verwenden. Sie finden die Klasse im Moodle-Lernraum.
Um Ihre Implementierung selbst zu testen, können sie in der Klasse `Rectangle` eine `main`-Methode schreiben, um verschiedene Szenarien zu überprüfen. Vergessen Sie nicht, Randfälle zu betrachten.

a) Erstellen Sie eine Klasse `Rectangle` mit den Attributen `x`, `y`, `width` und `height`.

b) Erstellen Sie die folgenden öffentlichen Methoden, um Objekte des Typs `Rectangle` erzeugen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren:

```
Rectangle(int xInput, int yInput, int widthInput, int heightInput)
Rectangle(int xInput, int yInput, int sidelengthInput)
Rectangle copy(Rectangle toCopy)
```

Beachten Sie dabei folgende Punkte:

- Der Konstruktor `Rectangle` mit vier Argumenten soll ein Rechteck erzeugen, dessen Attribute jeweils die von den entsprechenden Parametern angegebenen Werte haben.
- Der Konstruktor `Rectangle` mit drei Argumenten soll ein Quadrat erzeugen, dessen Höhe und Breite den Wert des Parameters `sidelengthInput` annehmen und dessen übrige Attribute jeweils die von den entsprechenden Parametern angegebenen Werte haben.
- Falls bei den ersten beiden Methoden eines der Argumente einen unzulässigen Wert hat, muss eine geeignete Fehlermeldung ausgegeben und direkt im Anschluss `return` ausgeführt werden. Zur Ausgabe einer Fehlermeldung kann die Methode `Utils.error(String msg)` genutzt werden.
- Die Methode `copy` soll ein Rechteck kopieren, d.h., es soll ein neues Rechteck zurückgeliefert werden, das die gleichen Attribut-Werte wie das Rechteck `toCopy` hat.

c) Erstellen Sie Selektoren, um die Koordinaten, die Breite und die Höhe eines Rechtecks setzen und auslesen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren. Falls ein Attribut auf einen unzulässigen Wert gesetzt werden soll, darf der Wert des Attributes nicht verändert werden. Stattdessen muss eine geeignete Fehlermeldung ausgegeben werden.

Hinweise:

- Um einen Fehler auszugeben, kann die Methode `Utils.error(String msg)` genutzt werden.

d) Erstellen Sie die folgenden öffentlichen Methoden. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren und begründen Sie Ihre Entscheidung kurz:

```
boolean areSquares(Rectangle... rectangles)
int area()
Rectangle intersection(Rectangle... rectangles)
```

Bei den in diesem Aufgabenteil geforderten Methoden muss der Aufrufer (und nicht Sie als Implementierer der Klasse `Rectangle`) sicherstellen, dass die Parameter, mit denen die Methoden aufgerufen werden, nicht den Wert `null` haben bzw. enthalten.

Beachten Sie dabei folgende Punkte:

- Die Methode `boolean areSquares(Rectangle... rectangles)` soll `true` zurückgegeben, falls jedes Rechteck in `rectangles` ein Quadrat ist.
- Die Methode `int area()` soll die Fläche des Rechtecks zurückgeben, auf dem sie aufgerufen wird.

- Die Methode `intersection(Rectangle... rectangles)` gibt das größte Rechteck zurück, das *vollständig* in *allen* als Argument übergebenen Rechtecken enthalten ist. Wenn `rectangles` leer ist, soll `null` zurückgegeben werden. Falls der Schnitt der Rechtecke leer ist, soll ebenfalls `null` zurückgegeben werden.
  Hinweis: Es empfiehlt sich, eine Hilfsmethode zu implementieren, die den Schnitt *zweier* Rechtecke berechnet.
  Beispiel: Wir betrachten die beiden Rechtecke, die mit den Aufrufen
  `new Rectangle(1,4,2,3)` und `new Rectangle(2,5,3,3)` erzeugt werden. Das ausgegebene Rechteck beim Aufruf von `intersection` soll ein Rechteck mit den Werten $2, 4, 1, 2$ zurückgegeben werden. Die Werte stehen jeweils in der Reihenfolge $x, y, width, height$.

  Hinweise:
  - Sie finden in der Klasse `Utils` zwei Methoden `min` und `max`, die das Minimum bzw. Maximum von beliebig vielen Werten des Typs `int` berechnen.

e) Erstellen Sie ebenfalls eine Implementierung für die öffentliche Methode

```
String toString()
```

Entscheiden Sie dabei selbst, ob Sie die Methode als statisch deklarieren. Die Methode `toString()` erstellt eine textuelle Repräsentation des aktuellen Rechtecks. Dies geschieht über die Eckpunkte des Rechtecks, die, beginnend bei der oberen linken Ecke, gegen den Uhrzeigersinn ausgegeben werden sollen.

Zum Beispiel stellt der String `(3|5),(3|1),(6|1),(6|5)` das Rechteck mit den Koordinaten 3 und 5, der Breite 3 und der Höhe 4 dar.

f) Dokumentieren Sie alle Methoden, die als `public` markiert sind, indem Sie die Implementierung mit `javadoc`-Kommentaren ergänzen. Diese Kommentare sollten eine allgemeine Erklärung der Methode sowie weitere Erklärungen jedes Parameters und des `return`-Wertes enthalten. Verwenden Sie innerhalb des Kommentars dafür die `javadoc`-Anweisungen `@param` und `@return`.

Benutzen Sie das Programm `javadoc`, um Ihre `javadoc`-Kommentare in das HTML-Format zu übersetzen. Überprüfen Sie mit einem Browser, ob das gewünschte Ergebnis generiert wurde. Falls `javadoc` Ihre Abgabe nicht kompiliert, werden **keine** Punkte vergeben.

Aufgabe 6 (Deck 4): (Codescape)

Lösen Sie die Missionen von Deck 4 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn Sie Ihre Lösung vor der einheitlichen Codescape Deadline am Samstag, den 22.01.2022, um 23:59 Uhr abschicken.