Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

In den Aufgaben 2 bis 4 sollen Sie Speicherzustände zeichnen. Angenommen wir haben folgenden Java Code:

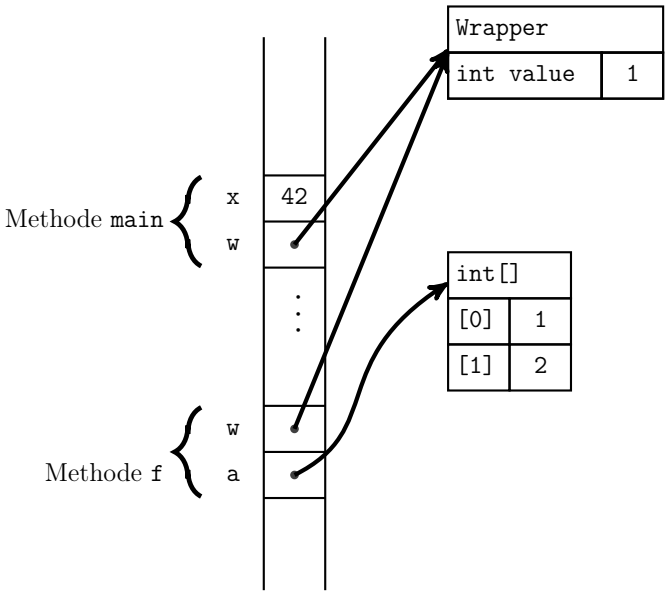
```
public class Wrapper {
    int value;
}
```

```
public class Main {
    public static void main(String[] args) {
        int x = 42;
        Wrapper w = new Wrapper();
        w.value = 0;
        f(w);
    }

    public static void f(Wrapper w) {
        int[] a = {1,2};
        w.value = 1;

        // Speicherzustand hier gezeichnet
    }
}
```

Dann sieht der Speicher an der markierten Stelle wie folgt aus:



Aufgabe 4 (Seiteneffekte): (7 + 7 + 6 = 20 Punkte)

Betrachten Sie das folgende Programm:

```
public class HSeiteneffekte {
        public static void main(String[] args) {
                HWrapper w1 = new HWrapper();
                HWrapper w2 = new HWrapper();

                w1.i = 0;
                w2.i = 3;

                int[] a = { 1, 2 };
```

```
                f(a, w1, w2);
                int[] b = {2*a[0],2*a[1]};
                f(b, w1);
                f(a);
        }

        public static void f(int[] a, HWrapper... ws) {
                if(ws.length==0){
                        a = new int[2];
                        a[0] = 8;
                        a[1] = 9;
                }else{
                        a[1] += a[0];
                        ws[ws.length-1].i = a[0];
                        ws[0].i += ws[ws.length-1].i;
                }
                //Speicherzustand jeweils hier zeichnen
        }
}

public class HWrapper {

    int i;

}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher (d.h. alle (implizit) im Programm vorkommenden Arrays (außer `args`) und alle Objekte sowie die zu dem Zeitpunkt existierenden Programmvariablen) am Ende jeder Ausführung der Methode `f` graphisch dar. Insgesamt sind also drei Speicherzustände zu zeichnen.

Aufgabe 7 (Einfache Klassen): (2 + 3 + 4 + 2 + 4 + 3 + 3 + 4 + 2 + 3 = 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.

a) Schreiben Sie jeweils 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 Klasse `Karte` mit je einem Attribut vom Typ `Farbe` und `Wert`. Schreiben 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 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 geändert haben. 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)`. Bei 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... karten)` in der Klasse `Karte` zu schreiben. Diese soll genau dann `true` zurückgeben, wenn mindestens eines der übergebenen `Karte`-Objekte dasjenige `Karte`-Objekt bedient, auf dem die Methode aufgerufen wurde.

f) Ergänzen Sie die Klasse `Karte` um eine statische Methode `void druckeEinbahnBedienungen()`. Diese Methode soll alle Paare von unterschiedlichen `Karte`-Objekten durchgehen und für jedes Paar (`k1`,`k2`) eine Meldung ausgeben, das folgende Bedingung erfüllt: Es gilt `k1.bedient(k2)`, aber nicht `k2.bedient(k1)`. Die Meldung soll mit `System.out.println` ausgegeben werden und folgende Form haben: `HERZBUBE bedient KAROKOENIG, aber KAROKOENIG nicht HERZBUBE.`

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 seine prozentuale Siegesquote, wobei hierbei auch die ersten Nachkommastellen interessant sind. 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` um eine statische Methode `Spieler besterSpieler(Spieler... club)`. Bei Aufruf dieser Methode mit einem oder mehreren Argumenten soll von diesen dasjenige `Spieler`-Objekt mit der höchsten Siegesquote zurückgegeben werden. Gibt es mehrere Spieler mit der gleichen (höchsten) Siegesquote, soll von diesen der erste übergebene Spieler zurückgegeben werden.

i) Ergänzen Sie die Klasse `Spieler` um eine Methode `void kannBedienen(Karte k)`. Mit Hilfe der Methode `System.out.println` soll eine Meldung der Form `"Gabi kann bedienen!"` oder `"Gabi kann nicht bedienen!"` ausgegeben werden, die anzeigt, ob der Spieler mit seiner Kartenhand das Karte-Objekt `k` bedienen kann. Anstelle von `"Gabi"` sollte der Name des Spielers stehen. Nutzen Sie zur Generierung dieser Ausgabe den `?`-Operator.

j) Ergänzen Sie die Klasse `Spieler` um eine `main`-Methode mit der bekannten Signatur. Erstellen Sie in dieser zuerst vier `Spieler`-Objekte für die Spieler Elisabeth, Klaus, Helmut und Erwin mit den Siegesquoten 37,5%, 12,5%, 38,75% und 11,25%. Lassen Sie das Attribut `kartenhand` uninitialisiert. Speichern Sie dann in einer fünften `Spieler`-Variable unter Benutzung der entsprechenden Methode den besten der vier soeben erstellten Spieler. Erstellen Sie ein dreielementiges Karten-Array mit den drei Karten Herz-Sieben, Herz-Neun und Karo-König und weisen sie dieses dem Attribut `kartenhand` des besten Spielers zu. Nutzen Sie die passende Methode, um festzustellen, ob der beste Spieler die Karte Karo-Bube bedienen kann.

Aufgabe 8 (Deck 3): (Codescape)

Lösen Sie die Missionen von Deck 3 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 16.01.2021, um 23:59 Uhr abschicken.