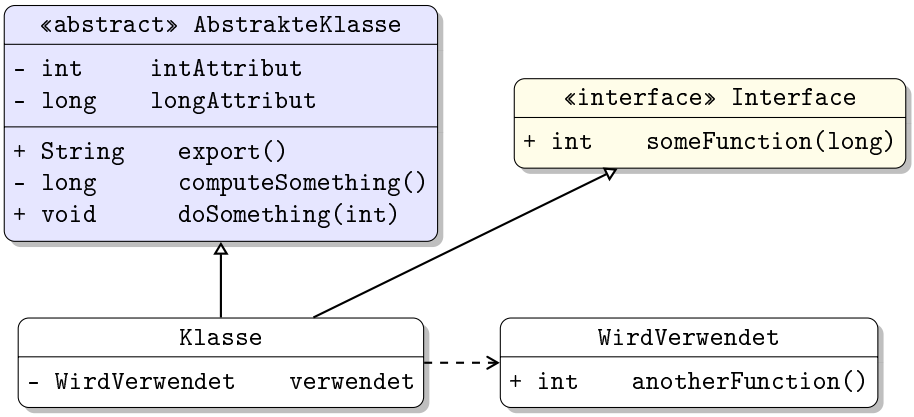
Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Klassenhierarchie): (10 Punkte)

In dieser Aufgabe soll der Zusammenhang verschiedener Getränke zueinander in einer Klassenhierarchie modelliert werden. Dabei sollen folgende Fakten beachtet werden:

- Jedes Getränk hat ein bestimmtes Volumen.
- Wir wollen Apfelsaft und Kiwisaft betrachten. Apfelsaft kann klar oder trüb sein.
- Alle Saftarten können auch Fruchtfleisch enthalten.
- Wodka und Tequila sind zwei Spirituosen. Spirituosen haben einen bestimmten Alkoholgehalt.
- Wodka wird häufig aromatisiert hergestellt. Der Name dieses Aromas soll gespeichert werden können.
- Tequila gibt es als silbernen und als goldenen Tequila.
- Ein Mischgetränk ist ein Getränk, das aus verschiedenen anderen Getränken besteht.
- Mischgetränke und Säfte kann man schütteln, damit die Einzelteile (bzw. das Fruchtfleisch) sich gleichmäßig verteilen. Sie sollen daher eine Methode `schuetteln()` ohne Rückgabe zur Verfügung stellen.
- In unserer Modellierung gibt es keine weiteren Getränke.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die Getränke. Notieren Sie keine Konstruktoren, Getter und Setter. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden.
Verwenden Sie hierbei die folgende Notation:



Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \longrightarrow\!\!\triangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist) und A `->` B, dass A den Typ B verwendet (z.B. als Typ eines Attributs oder in der Signatur einer Methode). Benutzen sie `+` und `-` um `public` und `private` abzukürzen.
Tragen Sie keine vordefinierten Klassen (`String`, etc.) oder Pfeile dorthin in ihr Diagramm ein.

Aufgabe 4 (Programmierung): (3.5 + 0.5 + 2 + 9 + 2 + 5.5 + 5.5 + 6 + 6 = 40 Punkte)

Ein Array bietet die Möglichkeit, einer begrenzten Anzahl natürlicher Zahlen einen Wert zuzuordnen. Als Schlüssel können bei Arrays also nur natürliche Zahlen genutzt werden. Oft ist es jedoch hilfreich, als Schlüssel Werte eines anderen Typs zu nutzen. Eine solche Datenstruktur nennt sich Map.
Beispielsweise kann in einer Map mit dem Schlüssel-Typ `String` und dem Wert-Typ `Integer` beliebigen `String`s je ein `Integer`-Wert zugeordnet werden.
In dieser Aufgabe wollen wir eine solche Map-Datenstruktur in `Java` implementieren, welche es uns außerdem erlaubt zu kontrollieren, ob nur lesend oder auch schreibend auf die Map zugegriffen werden kann.

Hinweise:

- (Pflicht) Nutzen Sie an geeigneten Stellen Modifizierer wie `static`, `private`, `protected` und `public`.
- (Pflicht) Wenn ein Typ Typparameter erwartet, so nutzen Sie ihn nicht ohne Typparameter.
- (Optional) Nutzen Sie an geeigneten Stellen Modifizierer wie `@Override` und `final`.

a) Zunächst erstellen wir eine Klasse, um einzelne Zuordnungen von Schlüsseln zu Werten speichern zu können.

Erstellen Sie die Klasse `Entry`, welche zwei generische Typparameter erhält. Der erste Typparameter, `K`, stellt den Typ der Schlüssel dar, der zweite, `V`, den Typ der Werte.

Die Klasse soll ein `K`-Attribut `key` haben und ein `V`-Attribut `value`. Die Klasse soll über einen Konstruktor verfügen, welcher die beiden Attribute entsprechend initialisiert. Die Klasse soll außerdem für beide Attribute einen Getter zur Verfügung stellen.

b) Erstellen Sie die Klasse `UnknownKeyException`, welche von `Exception` erbt.

c) Nun erstellen wir ein Interface, um lesend auf eine Map zugreifen zu können.

Erstellen Sie das Interface `ReadableMap`, welches ebenfalls die beiden generischen Typparameter `K` und `V` für den Typ der Schlüssel und den Typ der Werte erhält.

Das Interface hat eine Methode `getOrThrow`, die einen `K`-Parameter `key` erhält und einen `V`-Wert zurückgibt. Außerdem sollen Methoden, die `getOrThrow` implementieren, eine `UnknownKeyException` werfen, falls dem übergebenen Schlüssel kein Wert zugeordnet ist.

d) Nun erstellen wir eine abstrakte Klasse, welche ein Array von Schlüssel-Wert-Zuordnungen hält und darauf lesenden Zugriff gewährt. Diese Klasse ist nur deswegen abstrakt, da von ihr keine Objekte erzeugt werden sollen. Wir implementieren anschließend zwei Unterklassen, von denen tatsächlich Objekte erstellt werden können.

Erstellen Sie die abstrakte Klasse `AbstractReadableMap`, welche ebenfalls die beiden generischen Typparameter `K` und `V` für den Typ der Schlüssel und den Typ der Werte erhält und das Interface `ReadableMap<K, V>` implementiert.

Die Klasse soll ein Attribut `entries` vom Typ `Entry<K, V>[]` haben. Dieses Attribut wird durch einen Konstruktor initialisiert, welcher einen Parameter vom Typ `Entry<K, V>[]` erhält, dieses Array kopiert und das kopierte Array dem Attribut zuweist. Das Array wird deshalb einmal kopiert, damit diese Klasse die einzige ist, welche eine Referenz auf das Array hält, welches in ihrem Attribut gespeichert ist. So ist ausgeschlossen, dass das Array im Attribut von außen geändert werden kann. Außerdem soll ein weiterer Konstruktor ohne Parameter implementiert werden, welcher das Attribut mit einem Array der Länge 10 initialisiert.

Hinweise:

- (Info) Leider ist es nicht so einfach, mit Arrays eines generischen Typs zu arbeiten. Daher haben wir die Klasse `GenericArrayHelper` im Moodle-Lernraum zur Verfügung gestellt, um dies für Sie zu vereinfachen.
- (Pflicht) Nutzen Sie die Methode `<T> T[] copyArray(T[] array)` der Klasse `GenericArrayHelper`, um das übergebene Array zu kopieren. Das zurückgegebene Array ist also gleich groß wie das übergebene Array und hat dieselben Einträge, wurde jedoch neu angelegt.

- (Pflicht) Nutzen Sie die Methode `<K, V> Entry<K, V>[] newEntryArrayOfSize(int size)` der Klasse `GenericArrayHelper`, um ein Array der Länge `size` mit Einträgen vom Typ `Entry<K, V>` anzulegen.

Implementieren Sie die Methode `getOrThrow` des Interfaces `ReadableMap`. Laufen Sie dazu mit einer `foreach`-Schleife über das Attribut `entries`. Falls Sie dabei einen Arrayeintrag finden, welcher nicht `null` ist und dessen Schlüssel dem Parameter `key` entspricht, so geben Sie den Wert dieses Arrayeintrags zurück. Falls Sie keinen solchen Arrayeintrag finden, so werfen Sie eine `UnknownKeyException`.

Hinweise:

- (Pflicht) Nutzen Sie die Methode `equals`, um Schlüssel miteinander zu vergleichen.

e) Nun erstellen wir ein Interface, um schreibend auf eine Map zugreifen zu können.

Erstellen Sie das Interface `WritableMap`, welches ebenfalls die beiden generischen Typparameter `K` und `V` für den Typ der Schlüssel und den Typ der Werte erhält. Dieses Interface erweitert das Interface `ReadableMap<K, V>`, da bei schreibendem Zugriff auch automatisch ein lesender Zugriff möglich sein soll.

Das Interface hat die Methode `put`, welche einen `K`-Parameter `key` und einen `V`-Parameter `value` erhält und nichts zurückgibt.

f) Wir erstellen nun eine Klasse, welche lesenden und schreibenden Zugriff auf eine Map bietet.

Erstellen Sie die Klasse `MutableMap`, welche ebenfalls die beiden generischen Typparameter `K` und `V` für den Typ der Schlüssel und den Typ der Werte erhält und welche eine Unterklasse von `AbstractReadableMap<K, V>` ist und das Interface `WritableMap<K, V>` implementiert.

Implementieren Sie die Methode `put` des Interfaces `WritableMap`. Laufen Sie dazu mit einer `for`-Schleife über das Attribut `entries` der Klasse `AbstractReadableMap`. Damit Sie auf dieses Attribut aus einer Subklasse zugreifen können, sollte dieses als `protected` deklariert werden.

Hinweise:

- (Pflicht) Nutzen Sie die Methode `equals`, um Schlüssel miteinander zu vergleichen.

Falls Sie einen Arrayeintrag finden, welcher `null` ist oder dessen Schlüssel dem Parameter `key` entspricht, so überschreiben Sie diesen Arrayeintrag mit einem neu erstellten `Entry`-Objekt, welches den Parameter `key` als Schlüssel enthält und den Parameter `value` als Wert und beenden Sie anschließend die Ausführung der Methode mit folgender Anweisung:
`return;`

Falls Sie keinen solchen Arrayeintrag finden, so benötigen wir ein größeres Array. Erstellen Sie dazu ein Array doppelter Größe, kopieren Sie die bisherigen Arrayeinträge in das neue Array und weisen Sie dieses neue Array dem Attribut `entries` der Klasse `AbstractReadableMap` zu. Fügen Sie anschließend an der ersten freien Stelle im neuen Array ein neues `Entry`-Objekt ein, welches den Parameter `key` als Schlüssel und den Parameter `value` als Wert enthält.

Hinweise:

- (Info) Wie in Teil d) erwähnt, ist es leider nicht so einfach, mit Arrays eines generischen Typs zu arbeiten. Daher haben wir die Klasse `GenericArrayHelper` im Moodle-Lernraum zur Verfügung gestellt, um dies für Sie zu vereinfachen.
- (Pflicht) Nutzen Sie die Methode `<T> T[] copyArrayWithIncreasedSize(T[] array, int newSize)` der Klasse `GenericArrayHelper`, um ein neues Array der Länge `newSize` mit Einträgen vom Typ `T` zu erstellen und alle Einträge aus dem Parameter `array` in das neue Array zu kopieren. Die Methode gibt das neue Array zurück.

g) Bisher haben wir ein Interface für den lesenden Zugriff erstellt und eins für den schreibenden Zugriff. Es ist also bereits möglich, einer Methode, welche nur lesenden Zugriff benötigt, nur lesenden Zugriff zu gestatten. So können wir sicher sein, dass diese Methode keine Zuordnungen in der Map ändert. Es ist aber weiterhin möglich, dass sich Zuordnungen in einer Map vom Typ `ReadableMap` ändern, beispielweise, wenn das Objekt tatsächlich vom Typ `MutableMap` ist und an einer anderen Stelle noch eine Referenz vom Typ `WritableMap` auf dasselbe Objekt besteht. Eine Referenz vom Typ `ReadableMap` verhindert also nur, dass diese eine Referenz Zuordnungen in der Map ändert. Dies ist oft praktisch, jedoch ist

es manchmal notwendig, Änderungen vollständig auszuschließen. Daher erstellen wir nun eine Klasse, welche dies ermöglicht.

Erstellen Sie die Klasse `ImmutableMap`, welche ebenfalls die beiden generischen Typparameter `K` und `V` für den Typ der Schlüssel und den Typ der Werte erhält und welche eine Unterklasse von `AbstractReadableMap<K, V>` ist.

Die Klasse hat einen Konstruktor, welcher einen Parameter vom Typ `Entry<K, V>[]` erhält und diesen an den `super`-Konstruktor übergibt.

Markieren Sie diese Klasse als `final`, indem Sie vor das Schlüsselwort `class` das Schlüsselwort `final` schreiben. Dies führt dazu, dass keine andere Klasse diese Klasse als ihre Oberklasse definieren kann.

Wenn wir nun einen Wert vom Typ `ImmutableMap` haben, so wissen wir, dass es tatsächlich ein Objekt vom Typ `ImmutableMap` ist und nicht von einem Untertyp, denn es kann keine Untertypen von `ImmutableMap` geben. Da nun aber `ImmutableMap` keine Möglichkeit bietet, die Schlüssel-Wert-Zuordnungen zu ändern, ist klar, dass diese Zuordnungen sich nach dem Erstellen des `ImmutableMap`-Objekts nicht mehr ändern können. Die Zuordnungen können also nur einmalig beim Aufruf des Konstruktors festgelegt werden.

Erstellen Sie in dem Interface `ReadableMap` die Methode `asImmutableMap`, welche keine Parameter erhält und einen Wert vom Typ `ImmutableMap<K, V>` zurückgibt.

Implementieren Sie die Methode `asImmutableMap` aus dem Interface `ReadableMap` in der Klasse `AbstractReadableMap`. Rufen Sie dazu den Konstruktor von `ImmutableMap` auf, übergeben Sie das Attribut `entries` und geben Sie das so erstellte `ImmutableMap`-Objekt zurück.

h) Bei einem Array ist klar, welche Indizes es gibt, sobald man die Größe kennt. Bei einer Map ist jedoch zunächst unklar, welchen Schlüsseln ein Wert zugeordnet ist. Daher erstellen wir nun eine Methode, um die Schlüssel einer Map auszulesen.

Erstellen Sie in dem Interface `ReadableMap` die Methode `keysAsSet`, welche keine Parameter erhält und einen Wert vom Typ `Set<K>` zurückgibt.

Implementieren Sie die Methode `keysAsSet` aus dem Interface `ReadableMap` in der Klasse `AbstractReadableMap`. Erstellen Sie dazu ein neues `HashSet` und fügen Sie dort alle Schlüssel der aktuellen Map ein, etwa indem Sie mit einer `foreach`-Schleife über das `entries`-Attribut laufen und die Schlüssel aller nicht-leeren Arrayeinträge im `HashSet` ablegen. Geben Sie anschließend das erstellte `HashSet` zurück.

Hinweise:

- In dieser Aufgabe nutzen wir die von `Java` vorgegebenen Typen `java.util.Set` und `java.util.HashSet`. Dabei ist `Set<T>` ein Interface, welches von der Klasse `HashSet<T>` implementiert wird. Die Klasse `HashSet<T>` besitzt einen parameterlosen Konstruktor zur Erzeugung eines neuen leeren `HashSet`-Objekts. In `Set<T>` existiert die Methode `add(T t)`, welche den Wert `t` in das aktuelle `Set` einfügt.

i) Nun wollen wir unsere Map nutzen.

Erstellen Sie die Klasse `Launcher` mit einer `main`-Methode und zwei weiteren Methoden.

Die Methode `putEntries` erhält einen Parameter vom Typ `WritableMap<String, Integer>` und ruft darauf dreimal die Methode `put` auf, um folgende Zuordnungen zu erstellen:

- Dem `String`-Wert `"sizeInMB"` soll der `Integer`-Wert 42 zugeordnet werden.
- Dem `String`-Wert `"version"` soll der `Integer`-Wert 4 zugeordnet werden.
- Dem `String`-Wert `"yearOfRelease"` soll der `Integer`-Wert 2015 zugeordnet werden.

Die Methode `printEntries` erhält einen Parameter vom Typ `ReadableMap<String, Integer>` und gibt alle Schlüssel-Wert-Zuordnungen der übergebenen Map aus. Dazu wird zunächst die Methode `keysAsSet` auf der übergebenen Map aufgerufen und über den Rückgabewert mit einer `foreach`-Schleife gelaufen. In der Schleife wird die Methode `getOrThrow` mit jedem gefundenen Schlüssel `key` auf der übergebenen Map aufgerufen, um den `Integer`-Wert `value` zu erhalten, welcher dem Schlüssel zugeordnet ist. Diese Schlüssel-Wert-Zuordnung wird in folgender Form auf der Konsole ausgegeben:
`key: value`

Obwohl wir eindeutig nur für existierende Schlüssel die Methode `getOrThrow` aufrufen, müssen wir eine Fehlerbehandlung durchführen, denn die Methode `getOrThrow` ist mit `throws UnknownKeyException` deklariert. Erstellen Sie einen `try-catch`-Block, um diesen Fehler zu fangen. Geben Sie im `catch`-Block eine Nachricht auf der Konsole aus, welche beschreibt, dass dieser Fehler aufgetreten ist.

Die `main`-Methode arbeitet wie folgt. Zunächst wird einer Variable `map` vom Typ `MutableMap<String, Integer>` ein neues `MutableMap`-Objekt zugewiesen.

Anschließend wird die Methode `putEntries` mit `map` als Parameter aufgerufen. Da der Parameter der Methode `putEntries` vom Typ `WritableMap` ist, fordert diese Methode schreibenden Zugriff auf die `map` und erhält diesen auch, denn eine `MutableMap` ist auch immer eine `WritableMap`.

Anschließend wird die Methode `printEntries` mit `map` als Parameter aufgerufen. Da der Parameter der Methode `printEntries` vom Typ `ReadableMap` ist, fordert diese Methode nur lesenden Zugriff auf die `map` und erhält diesen auch, denn eine `MutableMap` ist auch immer eine `ReadableMap`.

Nun wird aus der `map` eine `ImmutableMap` mit den selben Einträgen erzeugt. Dazu wird auf `map` die Methode `asImmutableMap` aufgerufen und das Ergebnis in einer Variable `immutableMap` vom Typ `ImmutableMap<String, Integer>` abgelegt.

Anschließend wird die Methode `printEntries` mit `immutableMap` als Parameter aufgerufen. Da der Parameter der Methode `printEntries` vom Typ `ReadableMap` ist, fordert diese Methode nur lesenden Zugriff auf die `immutableMap` und erhält diesen auch, denn eine `ImmutableMap` ist auch immer eine `ReadableMap`.

Es ist jedoch nicht möglich, die Methode `putEntries` mit `immutableMap` als Parameter aufzurufen. Da der Parameter der Methode `putEntries` vom Typ `WritableMap` ist, fordert diese Methode schreibenden Zugriff auf die `immutableMap`. Dies ist jedoch nicht möglich, denn eine `ImmutableMap` kann nicht gleichzeitig eine `WritableMap` sein.

Die `main`-Methode sollte Folgendes ausgeben, wobei die Reihenfolge der Zeilen abweichend sein kann:

```
yearOfRelease: 2015
sizeInMB: 42
version: 4
yearOfRelease: 2015
sizeInMB: 42
version: 4
```

Hinweise:

- (Info) Hier sehen wir den Grund, warum es überhaupt Hüllklassen wie `Integer` und `Long` für die primitiven Typen `int` und `long` gibt. Der Grund ist, dass an manchen Stellen Datentypen verwendet werden müssen, die nicht primitiv sind. Beispielsweise können primitive Typen nicht als generische Typparameter genutzt werden. Ebenso kann man keinen Wert eines primitiven Typs an Stellen verwenden, wo ein Objekt vom Typ `Object` erwartet wird. Anstelle der primitiven Typen muss hier also immer die Hüllklasse genutzt werden.

Aufgabe 5 (Deck 7): (Codescape)

Schließen Sie das Spiel Codescape ab, indem Sie die letzten Missionen von Deck 7 lösen. Genießen Sie anschließend das Outro. Dieses Deck enthält keine für die Zulassung relevanten Missionen.

Hinweise:

- Es gibt drei verschiedene Möglichkeiten wie die Story endet, abhängig von Ihrer Entscheidung im finalen Raum.
- Verraten Sie Ihren Kommilitonen nicht, welche Auswirkungen Ihre Entscheidung hatte, bevor diese selbst das Spiel abgeschlossen haben.