Prof. Dr. J. Giesl
M. Hark, J. Hensel, D. Korzeniewski

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Dienstag, den 19.12.2017 um 08:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor oder unmittelbar vor Beginn in der **Globalübung** abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** schicken Sie sie per **E-Mail** vor Dienstag, dem 19.12.2017 um 08:00 Uhr an Ihre Tutorin/Ihren Tutor.
  Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird und über eine geeignete `main`-Methode verfügt, um mit `java` ausgeführt zu werden. Verändern Sie außerdem den gegebenen Code nur an den angegebenen Stellen. Ansonsten werden keine Punkte vergeben.
- Benutzen Sie in ihrem Code keine Umlaute, auch nicht in Strings und Kommentaren. Diese führen oft zu Problemen, da diese Zeichen auf verschiedenen Betriebssystemen unterschiedlich behandelt werden. Dadurch kann es dazu führen, dass Ihr Programm bei Ihrer Tutorin/Ihrem Tutor bei Verwendung von Umlauten nicht mehr von `javac` akzeptiert wird.
- Halten Sie sich beim Lösen von Programmieraufgaben an die auf der Website zur Vorlesung verfügbaren Codekonventionen. Verstöße, die zu unleserlichem Code führen, können zu Punktabzug führen.

Tutoraufgabe 1 (Collections):

In dieser Aufgabe geht es um die Implementierung einer Datenstruktur für Mengen, welche in das bestehende Collections Framework eingebettet werden soll. Sie benötigen dafür die Klassen `EmptySet`, `AddSet`, `RemoveSet`, `FunctionalSet`, `SimpleFunctionalSet` und `Main`, welche Sie als `.java` Dateien von unserer Webseite herunterladen können.
Die in dieser Aufgabe zu betrachtende Mengenstruktur basiert auf einer Liste von Einfüge- (`Add`) und Löschoperationen (`Remove`) mit jeweils einem Element, die vom Ausgangspunkt (起点) einer leeren Menge (`Empty`) angewendet werden. Zum Beispiel lässt sich die Menge $\{1, 2, 3\}$ als die Liste `Add 3, Add 2, Add 1, Empty` darstellen. Will man nun das Element 2 aus der Menge löschen, so entfernt man nicht das zweite Element aus der Liste, sondern fügt ein weiteres `Remove` Element hinzu und erhält `Remove 2, Add 3, Add 2, Add 1, Empty`. Auf diese Weise erhält man eine Datenstruktur, bei der niemals Objekte entfernt werden (mit Ausnahme der `clear` Methode, welche die Liste wieder auf `Empty` setzen soll).

a) Die vorgegebene Klasse `FunctionalSet` implementiert bereits das `Set` Interface bis auf die `iterator` Methode. Implementieren Sie diese Methode. Implementieren Sie dazu eine weitere generische Klasse `FunctionalSetIterator<E>`, welche das Interface `Iterator<E>` aus dem Package `java.util` implementiert. Schlagen Sie für die zu implementierenden Methoden `hasNext`, `next` und `remove` die Funktionalitäten in der Java API für das Interface `Iterator` nach (die `remove` Operation soll durch Ihren Iterator unterstützt werden, die Methode `forEachRemaining` brauchen Sie hingegen nicht zu implementieren). Dies betrifft insbesondere auch die durch diese Methoden zu werfenden Exceptions.

b) Implementieren Sie in der Klasse `FunctionalSet` eine Methode `E min(java.util.Comparator<E> comp)`, die das kleinste in der Menge gespeicherte Element zurückliefert. Die Ordnung, die zum Vergleich zweier Elemente verwendet wird, ist durch den `Comparator comp` festgelegt. Wenn die Menge leer ist, soll die Methode eine `MinimumOfEmptySetException` werfen. Implementieren Sie zu diesem Zweck eine Klasse `MinimumOfEmptySetException`, die von `java.lang.RuntimeException` erbt.

c) Sie können die `main` Methode der Klasse `Main` nutzen, um Ihre Implementierung zu testen. Allerdings stürzt diese ab, wenn Sie z.B. `add k` oder `remove k` eingeben, da `k` keine Zahl ist und das Parsen von `k` folglich mit einer `java.lang.NumberFormatException` scheitert. Die Methode stürzt ebenfalls ab, wenn Sie `min` eingeben, ohne vorher Elemente zu der Menge hinzuzufügen (indem Sie z.B. `add 2` eingeben). In diesem Fall ist der Grund eine `MinimumOfEmptySetException`. Fangen Sie diese Exceptions mit `try-catch`, um Programmabstürze zu verhindern und geben Sie stattdessen geeignete Fehlermeldungen aus.

Aufgabe 2 (Collections): (1+2+5+6+1+3+1+3 = 22 Punkte)

Auch in dieser Aufgabe geht es um die Implementierung von Datenstrukturen für Mengen.

a) Implementieren Sie eine nicht-abstrakte Klasse `mySets.MySetElement` (d.h., die Klasse mit Namen `MySetElement` soll sich im Paket `mySets` befinden). Diese soll einen Eintrag in einer Menge repräsentieren, die als einfach verkettete Liste implementiert ist. Zu diesem Zweck soll sie einen Typparameter `T` haben, der dem Typ der in der Menge gespeicherten Elemente entspricht. Außerdem soll sie über ein Attribut `next` vom Typ `MySetElement<T>` und ein Attribut `value` vom Typ `T` verfügen. Schreiben Sie auch einen geeigneten Konstruktor für die Klasse. Die Klasse `MySetElement` soll nur innerhalb des Pakets `mySets` sichtbar sein.

b) Implementieren Sie eine nicht-abstrakte Klasse `mySets.MySetIterator`, die das Interface `Iterator` aus dem Paket `java.util` implementiert. Sie soll einen Typparameter `T` haben, der dem Typ der Elemente entspricht, über die iteriert wird. Außerdem soll sie über ein Attribut `current` vom Typ `MySetElement<T>` verfügen, dessen initialer Wert dem Konstruktor als einziges Argument übergeben wird. Die Methode `hasNext` soll `true` zurückgeben, wenn `current` nicht `null` ist. Die Methode `next` soll das in `current` gespeicherte Objekt vom Typ `T` zurückliefern und `current` auf das nächste `MySetElement` setzen, wenn `current` nicht `null` ist. Es soll eine `java.util.NoSuchElementException` geworfen werden, falls `current == null` gilt. Die Methode `remove` (die in der Java-API als optional gekennzeichnet ist) soll eine `java.lang.UnsupportedOperationException` werfen. Die Methode `forEachRemaining` brauchen Sie nicht zu implementieren. Die Klasse `MySetIterator` soll nur innerhalb des Pakets `mySets` sichtbar sein.

c) Implementieren Sie eine abstrakte Klasse `mySets.MyAbstractSet`, die eine Menge repräsentiert und einen Typparameter `T` hat, der dem Typ der in der Menge gespeicherten Elemente entspricht. Außerdem soll `MyAbstractSet` das Interface `java.lang.Iterable` implementieren. Die Methode `iterator` soll dabei eine geeignete Instanz von `MySetIterator` zurückliefern, die anderen Methoden aus `Iterable` müssen Sie nicht überschreiben. Darüber hinaus soll `MyAbstractSet` alle Methoden des Interfaces `java.util.Set` mit Ausnahme von

- allen Methoden, die bereits in `java.lang.Object` implementiert sind,
- allen Methoden, die eine Default-Implementierung haben,
- allen statischen Methoden und
- allen Methoden, die in der zugehörigen Dokumentation[1] als optional gekennzeichnet sind

auf sinnvolle Art und Weise implementieren. Für die beiden `toArray`-Methoden können Sie konkrete Implementierungen angeben, die nur eine `java.lang.UnsupportedOperationException` werfen.

Die Klasse `MyAbstractSet` soll eine Menge als einfach verkettete Liste von `MySetElements` implementieren. Zu diesem Zweck soll sie über ein Attribut `head` vom Typ `MySetElement<T>` verfügen, dessen initialer Wert dem Konstruktor als einziges Argument übergeben wird. Die Klasse `MyAbstractSet` soll nur innerhalb des Pakets `mySets` sichtbar sein.

d) Implementieren Sie eine nicht-abstrakte Klasse `mySets.MyMutableSet`, die von `MyAbstractSet` erbt. Diese soll das Interface `java.util.Set` implementieren. Folglich müssen nun alle optionalen Methoden des Interfaces `java.util.Set` implementiert werden, da diese von `MyAbstractSet` nicht zur Verfügung

[1] `https://docs.oracle.com/javase/9/docs/api/java/util/Set.html`

gestellt werden. Die optionale Methode `retainAll` dürfen Sie mit einer Methode überschreiben, die nur eine `java.lang.UnsupportedOperationException` wirft. Da Instanzen von `MyMutableSet` Mengen repräsentieren, müssen alle Methoden, die Elemente in die Menge einfügen, sicherstellen, dass die zugrunde liegende Liste duplikatfrei bleibt. Dabei werden zwei Objekte `x` und `y` als "gleich" betrachtet, falls `x.equals(y)` den Wert `true` zurückliefert. Falls ein Element in ein `MyMutableSet` eingefügt werden soll, das bereits enthalten ist, soll die Liste unverändert bleiben. Der einzige Konstruktor der Klasse `MyMutableSet` nimmt keine Argumente entgegen und ruft den (einzigen) Konstruktor der Oberklasse `MyAbstractSet` mit dem Argument `null` auf. Die Klasse `MyMutableSet` soll öffentlich sichtbar sein.

e) Entwerfen Sie ein Interface `mySets.MyMinimalSet`. Implementierungen dieses Interfaces sollen unveränderliche Mengen repräsentieren. Das Interface soll die folgende Funktionalität zur Verfügung stellen:

- Es soll einen Typparameter `T` haben, der dem Typ der gespeicherten Elemente entspricht.
- Es soll eine Methode anbieten, um zu überprüfen, ob ein gegebenes Element Teil der Menge ist.
- Es soll eine Methode `void addAllTo(Collection<T> col)` zur Verfügung stellen, die alle Elemente des `MyMinimalSet`s zu der als Argument übergebenen `Collection` hinzufügt.
- Es soll das Interface `java.lang.Iterable` erweitern.

Das Interface `MyMinimalSet` soll öffentlich sichtbar sein.

f) Implementieren Sie eine nicht-abstrakte Klasse `mySets.MyImmutableSet`, die von `MyAbstractSet` erbt und das Interface `MyMinimalSet` implementiert. Diese soll einen Typparameter `T` haben, der dem Typ der gespeicherten Elemente entspricht. Ihr einziger Konstruktor nimmt den initialen Wert für das Attribut `head` der Oberklasse `MyAbstractSet` als Argument entgegen und ruft den (einzigen) Konstruktor von `MyAbstractSet` entsprechend auf. Die Klasse `MyImmutableSet` soll nur innerhalb des Pakets `mySets` sichtbar sein.

g) Implementieren Sie in der Klasse `MyMutableSet` eine öffentliche Methode `freezeAndClear()` mit dem Rückgabetyp `MyMinimalSet<T>`. Diese soll eine Instanz von `MyImmutableSet` zurückliefern, die die gleiche Menge repräsentiert wie `this`. Außerdem soll sie das Attribut `this.head` auf `null` setzen. Das heißt, die Methode `freezeAndClear` erstellt aus einem `MyMutableSet` ein `MyImmutableSet`, leert die ursprüngliche Menge und liefert das neu erstellte `MyImmutableSet` zurück.

h) Wie Sie bereits in den vorherigen Teilaufgaben gesehen haben, sind alle Methoden, die Elemente zu `Collection`s hinzufügen, als optional gekennzeichnet. Tatsächlich gibt es in der Java-Standardbibliothek einige `Collection`s, die das Hinzufügen von Elementen nicht unterstützen. Diese `Collection`s werfen typischerweise eine `java.lang.UnsupportedOperationException`, wenn eine nicht unterstützte Methode aufgerufen wird. Dies kann dazu führen, dass die Methode `addAllTo` der Klasse `MyImmutableSet` mit einer `UnsupportedOperationException` scheitert. Wir wollen dieses Problem nun explizit machen, indem die Methode `addAllTo` ggf. anstelle einer `UnsupportedOperationException`, welche eine `RuntimeException` ist, eine normale `Exception` wirft, die in der Methodensignatur deklariert werden muss. Implementieren Sie dazu eine nicht-abstrakte Klasse `mySets.UnmodifiableCollectionException` als Unterklasse von `java.lang.Exception`. Ändern Sie die Signatur von `MyMinimalSet.addAllTo` so, dass eine `UnmodifiableCollectionException` geworfen werden darf. Fangen Sie in der Methode `addAllTo` der Klasse `MyImmutableSet` evtl. auftretende `UnsupportedOperationException`s und werfen Sie stattdessen eine `UnmodifiableCollectionException`.

Hinweise:

- Beachten Sie, dass das Paket `mySets` ausschließlich dazu dient, die in der Aufgabenstellung beschriebenen Implementierungen von Mengen zur Verfügung zu stellen. Diese sind eng miteinander verknüpft (z.B. verwenden sowohl `MyMutableSet` als auch `MyImmutableSet` die Klasse `MySetElement`). Daher bietet es sich in dieser Aufgabe an, nicht alle Attribute als `private` zu deklarieren. Dies erhöht die Lesbarkeit des Codes, da direkt auf die entsprechenden Attribute zugegriffen werden kann. Dabei ist es kein Verstoß gegen das Prinzip der Datenkapselung, solange es Klassen außerhalb des Pakets `mySets` nicht möglich ist, auf nicht-private Attribute zuzugreifen.
- `Collection<?>` in der Signatur der Methoden `removeAll` und `retainAll` steht für eine `Collection` beliebiger Elemente, d.h., `removeAll` kann z.B. mit einer `Collection<Integer>`, aber auch mit einer `Collection<String>` als Argument aufgerufen werden.

- `Collection<? extends E>` in der Signatur der Methode `addAll` steht für eine `Collection` von Elementen eines beliebigen Subtyps von `E`. Wenn `F` ein Subtyp von `E` ist, kann `addAll` also sowohl mit einer `Collection<E>` als auch mit einer `Collection<F>` als Argument aufgerufen werden.