Prof. Dr. J. Giesl
S. Dollase, M. Hark, 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 oben links!**
- 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 Freitag, den 14.12.2018 um 12:00** 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 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 Freitag, den 14.12.2018 um 12:00 an Ihre Tutorin/Ihren Tutor. Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, 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.
- Für einige Programmieraufgaben benötigen Sie die Java Klasse `SimpleIO`. Diese können Sie auf der Website herunterladen.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden:
  https://codescape.medien.rwth-aachen.de/progra/
  Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Pakete, Module, Exceptions, Generics und 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 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. Die Methode `iterator` benötigt die generische Klasse `FunctionalSetIterator<E>`, welche das Interface `Iterator<E>` aus dem Package `java.util` implementiert. Schreiben Sie diese generische Klasse.

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.

d) Teilen Sie die Klassen aus dieser Aufgabe sinnvoll in Pakete und Module auf. Nur intern verwendete Klassen sollten dabei in einem eigenen Paket sein, das nicht exportiert wird. Alle Klassen, die ein Nutzer benötigt, müssen in einem Paket sein, das exportiert wird.

Aufgabe 2 (Pakete, Module, Exceptions, Generics und Collections): (1 + 2 + 5 + 8 + 1 + 5 + 3 + 1 = 26 Punkte)

Im Film "Monty Python and the Holy Grail"[1] wird an verschiedenen Stellen über Schwalben, ihre Fluggeschwindigkeit und die Frage diskutiert, ob sie als Zugvögel Kokosnüsse aus Afrika nach England gebracht haben könnten. In dieser Aufgabe werden wir uns damit beschäftigen, wie man einige dieser Zusammenhänge in Java darstellen könnte.
Schicken Sie die **Lösung zu dieser Aufgabe als zip-Datei** an Ihre Tutorin/Ihren Tutor, damit die **Ordnerstruktur erhalten bleibt!** Bei der Nutzung von Paketen und Modulen, müssen die Quellcode-Dateien in den richtigen (Unter-)Ordnern liegen. Andernfalls **compiliert Ihr Code nicht und wird mit 0 Punkten bewertet.**

Hinweise:

- Falls Sie zur Laufzeit Fehler mit der Beschreibung `java.lang.NoClassDefFoundError` erhalten, prüfen Sie, ob alle Klassen compiliert wurden und im richtigen Pfad liegen.

a) Schreiben Sie eine Klasse `Nut` in dem Paket `cargo`, um eine Nuss (zum Beispiel eine Kokosnuss) zu repräsentieren. Eine Nuss hat die Attribute `name` vom Typ `String` und `weight` vom Typ `int`. Beide Werte sollen auf einen beliebigen Wert außer `null` initialisiert werden. Schreiben Sie außerdem Selektoren zum Lesen und Schreiben beider Werte.

b) Schreiben Sie eine abstrakte Klasse `Swallow` im Paket `bird`, um eine allgemeine Schwalbe darzustellen. Jede Schwalbe kann ein Objekt vom Typ `Object` als Fracht tragen. Diese Fracht wird dem Konstruktor übergeben und kann mit der öffentlichen Methode `getCargo` abgefragt werden. Außerdem hat jede Schwalbe eine Methode `isLadden`, die `true` zurück gibt, falls die Fracht nicht `null` ist, und sonst `false`. Schließlich gibt es noch eine abstrakte Methode `getAirspeedVelocity`[2], die ein `int` zurückliefert und nur im gleichen Paket und aus Unterklassen genutzt werden darf.

c) Schreiben sie zwei Unterklassen der Klasse `Swallow`, nämlich `EuropeanSwallow` und `AfricanSwallow` im Paket `bird.swallows` mit entsprechenden Konstruktoren.

Die Airspeed Velocity einer europäischen Schwalbe ist $11\frac{m}{s}$, die einer afrikanischen Schwalbe ist $12\frac{m}{s}$. Die entsprechende Methode liefert nur den Betrag zurück, die Einheit können Sie ignorieren. Falls die Schwalbe mit einer `Nut` beladen ist, dann reduziert sich die Geschwindigkeit jedoch um den Betrag des Gewichts der Nuss. Falls sich dadurch ein negativer Wert ergeben würde, wird die Geschwindigkeit stattdessen auf 0 gesetzt. Eine europäische Schwalbe, die mit einer Nuss mit Gewicht 5 beladen ist, hätte

[1] Deutscher Titel: "Die Ritter der Kokosnuß" *(noch in alter Rechtschreibung)*
[2] Airspeed ist die Fluggeschwindigkeit relativ zur umgebenden Luft. Diese Größe ist wichtiger als die Geschwindigkeit über Grund, da sie maßgeblich für den Auftrieb ist.

also nur eine Airspeed Velocity von 6. Falls die Schwalbe mit einer Fracht beladen ist, die keine Nuss ist, so halbiert sich die Geschwindigkeit, wobei nach unten abgerundet wird.
Schreiben Sie hierzu eine geschützte Hilfsmethode in der Klasse `Swallow`, die gleiche Teile der beiden Implementierungen von `getAirspeedVelocity` enthält.

Schreiben Sie außerdem je eine statische Methode `createAfricanSwallow` und `createEuropeanSwallow` in der Klasse `Swallow`, die jeweils eine Fracht `cargo` vom Typ `Object` übergeben bekommen und ein `Swallow`-Objekt zurück geben. Die Methode soll ein neues Objekt der entsprechenden Klasse mit der Fracht `cargo` erstellen und zurück geben.

d) Schwalben leben und wandern in Schwärmen. Schreiben Sie im Paket `bird` eine generische Klasse `Flock` mit dem Typ-Parameter `S`, um einen Schwarm Schwalben zu repräsentieren. Ein Schwarm kann aber nicht aus anderen Dingen als aus Schwalben bestehen. Ein Schwarm besteht aus einer `List` von Schwalben. Diese Liste darf nur Objekte vom Typ `S` enthalten. Ein Schwarm afrikanischer Schwalben kann also keine europäischen Schwalben enthalten. Ein Schwarm allgemeiner Schwalben kann jedoch beide Schwalbenarten enthalten.

Erstellen Sie einen Konstruktor ohne Parameter sowie eine Methode `join`, die eine Schwalbe passenden Typs zum Schwarm hinzufügt. Die Klasse `java.util.ArrayList` ist eine Implementierung des Interfaces `List`, die sich für diese Aufgabe anbietet.

Ein Schwarm hat eine Methode `double getAverageCruiseAirspeedVelocity()`, um die durchschnittliche Airspeed Velocity zu berechnen. Die hinteren Schwalben fliegen im Windschatten der vorderen, daher müssen sie nicht gegen eventuellen Gegenwind anfliegen. Ziehen Sie also von der Airspeed Velocity der ersten Schwalbe 2 ab, ziehen Sie bei der zweiten Schwalbe 1 ab und berechnen Sie erst dann den Durchschnitt.

Schreiben Sie ein Interface `FlockInterface`, das von `Flock` implementiert wird und nur die öffentliche Methode `getAverageCruiseAirspeedVelocity` besitzt. Dieses Interface dient als gemeinsame Oberklasse aller generischen Varianten von `Flock`, mit allen wichtigen, vom Typparameter unabhängigen Methoden, und kann statt des Raw-Typs verwendet werden.[3]

e) Erstellen Sie eine Exception-Klasse `UnspecificQuestionException` im Paket `people`. Es soll sich um eine *checked* Exception handeln.

f) Erstellen Sie im Paket `people` eine Klasse `Troll`. Ein Troll kann verwirrt sein oder nicht und hat daher ein Attribut `confused`, das auf `false` initialisiert wird. Verwirrtheit sieht man einem Troll aber nicht an, daher gibt es für dieses Attribut keine öffentlichen Selektoren.

Ein Troll hat eine Methode `pass`, mit der man versuchen kann, an ihm vorbei zu gehen. Das funktioniert nur, wenn er verwirrt ist. Dann passiert in dieser Methode nichts. Andernfalls beendet der Troll das Programm mit Hilfe der Methode `java.lang.System.exit(-1)`.

Ein Troll hat eine Methode `askAboutAirspeedVelocity(Object)`. Falls es sich bei dem Objekt um ein Objekt vom Typ `FlockInterface` handelt, gibt die Methode einfach die durchschnittliche Airspeed Velocity zurück. Ist es ein `Swallow`-Objekt, erstellt der Troll einen Schwarm allgemeiner Schwalben vom Typ `Swallow` mit genau diesem einen Schwarmmitglied, nimmt die durschnittliche Airspeed Velocity, addiert 2 (um den Gegenwind herauszurechnen) und gibt das Ergebnis zurück.

Für alle anderen Objekte gibt die Methode 0 zurück, mit einer Ausnahme: Ist es ein String, der den Wert `"Swallow"`, `"Unladden Swallow"`, `"European Swallow"` oder `"African Swallow"` hat, so ist der Troll verwirrt und wirft eine `UnspecificQuestionException`.

g) Teilen Sie ihr Projekt in drei Module auf. Das Modul `cargo` enthält nur das Paket `cargo`. Das Modul `bird` enthält die Pakete `bird` und `bird.swallows`, stellt aber nur das erste Paket für andere Module zur Verfügung. Das Modul `people` enthält nur das Paket `people`.

h) Erweitern Sie die `main`-Methode in der Klasse `people.KingArthur` um einen Abschnitt, in dem der Troll eine Frage gestellt bekommt, die er nicht beantworten kann. Fangen Sie die auftretende Exception ab und passieren sie anschließend den Troll.

[3] Dies führt zu einem besseren Programmierstil als `instanceof` mit dem Raw-Typ zu verwenden.

Aufgabe 3 (Codescape): (Codescape)

Lösen Sie die Räume von **Deck 6** des Spiels Codescape.
Ihre Lösung für Räume dieses Codescape Decks wird nur dann für die Zulassung gezählt, wenn Sie die Lösung bis Freitag, den 14.12.2018 um 12:00 abschicken.