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

Tutoraufgabe 1 (Überblickswissen):

a) Sammeln Sie verschiedene "populäre" Java-Exceptions, z.B. in der Vorlesung eingeführte Exceptions oder anderswo her bereits bekannte Exceptions, und diskutieren Sie, ggf. an Beispielen, wann und warum die jeweiligen Exceptions auftreten.

b) In der Vorlesung haben Sie zwei Arten von allgemeinen Listen kennengelernt. Einmal können generische Typen benutzt werden, um eine solche allgemeine Liste zu realisieren. Die zweite Implementation hingegen benutzt Werte vom Typ `Object`. Was sind die Nachteile dieser Umsetzung?

c) Wofür benötigt ein*e Java-Entwickler*in Module und Pakete?

d) Woraus besteht das Collection-Framework[1]? Warum ist es in den meisten Fällen sinnvoll, ein solches Framework zu verwenden und nicht eigene Implementierungen?

Tutoraufgabe 2 (Collections, Exceptions und Generics):

In dieser Aufgabe geht es um die Implementierung einer Datenstruktur für Mengen, welche in das bestehende Collection-Framework eingebettet werden soll. Sie benötigen dafür die Klassen `EmptySet`, `AddSet`, `RemoveSet`, `FunctionalSet`, `SimpleFunctionalSet` und `Main`, welche Sie als `.java` Dateien im Moodle-Lernraum 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 teilweise das `Set` Interface. Ergänzen Sie die Klasse um einen Konstruktor `FunctionalSet()` und eine geeignete `toString`-Methode. Der Konstruktor soll eine leere Menge erzeugen. Erweitern Sie das Interface außerdem um sinnvoll implementierte Methoden `boolean add(E e)`, `boolean remove(Object o)`, `boolean addAll (Collection <? extends E > c)` und `boolean removeAll(Collection <?> c)`. Hierbei gibt der Rückgabewert an, ob die Methode die Datenstruktur verändert hat. Schreiben Sie zuletzt noch eine Methode `void clear()`, die eine entsprechende Menge `this` leert.

b) 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.

c) 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.

d) 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,

[1] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Collection.html`

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.

Tutoraufgabe 3 (Exceptions und Generics (Video)):

Diese Aufgabe behandelt erneut, wie bereits Aufgabe 4 auf dem vorherigen Blatt, einen Weihnachtsmarkt. Sie haben bereits auf Blatt 7 für Weihnachtsmärkte eine Klassenhierarchie entworfen. Auf diesem Blatt wird nun die tatsächliche Implementation behandelt. Verwenden Sie hierzu die Hilfsklassen `Zufall` und `SimpleIO`, die beide im Moodle-Lernraum zu finden sind.

- Ein Weihnachtsmarkt besteht aus verschiedenen Ständen. Ein Weihnachtsmarkt verfügt außerdem über eine Methode `run()`, die kein Ergebnis zurückgibt.
- Ein Stand kann entweder ein Weihnachtsartikelstand oder ein Lebensmittelstand sein. Jeder Stand hat eine*n Verkäufer*in, dessen Name von Interesse ist, und eine Anzahl von Besuchern*innen pro Stunde. Hierfür existiert sowohl ein Attribut `besucherProStunde` als auch eine Methode `berechneBesucherProStunde()`, um diese Anzahl neu zu berechnen. Ein Stand bietet außerdem die Methode `einzelkauf()`, welche den zu bezahlenden Preis (centgenau in Euro) zurückgibt.
- Ein Weihnachtsartikelstand hat eine Reihe an Artikeln.
- Ein Artikel hat einen Namen und einen Preis (centgenau in Euro).
- Ein Lebensmittelstand verkauft ein bestimmtes Lebensmittel.
- Ein Lebensmittel ist entweder ein Flammkuchen oder eine Süßware. Es bietet die Möglichkeit, über die Methoden `getPreisPro100g()` und `getName()`, den festen Preis pro 100 Gramm (centgenau in Euro) und den Namen abzurufen.
- Bei einer Süßware ist der Preis pro 100 Gramm (centgenau in Euro) und die Süßwarenart als String von Interesse.
- Bei einem Flammkuchen ist der Preis pro 100 Gramm (centgenau in Euro) von Interesse.
- Ein Süßwarenstand ist ein Lebensmittelstand.
- Im Gegensatz zu Flammkuchenständen, die einen festen Wasseranschluss benötigen, lassen sich Weihnachtsartikelstände und Süßwarenstände mit einer Methode `verschiebe(int)` ohne Rückgabe verschieben. Dies wird regelmäßig ausgenutzt, falls die Anzahl der Besucher*innen erhöht werden soll.

a) Klassenhierarchie: Siehe Blatt 7 Aufgabe 4

b) Implementieren Sie die Klassen entsprechend Ihrer Klassenhierarchie. Nutzen Sie dafür die Klassen `Zufall` und `SimpleIO` (aus dem Moodle-Lernraum). Fügen Sie jeder Klasse, falls notwendig, Getter und Setter sowie eine geeignete `toString`-Methode hinzu.

c) Die Klasse `Lebensmittelstand` soll einen generischen Typparameter `T` erhalten, welcher `Lebensmittel` oder eine Unterklasse von `Lebensmittel` sein kann. Das einzige Attribut `lebensmittel` der Klasse `Lebensmittelstand` soll vom Typ `T` sein. Der Konstruktor erhält einen Parameter vom Typ `T` und weist das Attribut entsprechend zu. Die Klasse `Suesswarenstand` ist Unterklasse von `Lebensmittelstand<Suessware>` und benötigt einen Konstruktor, welcher den `super`-Konstruktor mit einem `Suessware`-Objekt aufruft.

d) Initialisieren Sie die Attribute der Objekte Ihrer Klassen mithilfe von `Zufall.java` nach folgendem Schema:

- Der Konstruktur der Klasse `Weihnachtsmarkt` bekommt die Anzahl der Stände übergeben und legt ein Array mit entsprechend vielen Ständen an. Verwenden Sie die statische Methode `Zufall.zahl(int)` so, dass es jeweils etwa 33% Weihnachtsartikelstände, Süßwarenstände und Flammkuchenstände gibt. Hierbei gibt ein Aufruf `Zufall.zahl(i)` (für `i` größer 0) eine zufällige Zahl zwischen 0 und `i`−1 zurück. Ein Flammkuchenstand kann erzeugt werden, indem der `Lebensmittelstand`-Konstruktor mit einem `Flammkuchen`-Objekt aufgerufen wird.
- Der Name der Verkäuferin oder des Verkäufers eines Stands wird mit der statischen Methode `Zufall.name()` festgelegt.
- Die Anzahl der Besucher*innen pro Stunde bewegt sich zwischen 0 und 100 und soll mit der Methode `Zufall.zahl(int)` festgelegt werden. Die einzige Ausnahme bilden Weihnachtsartikelstände, bei denen sich die Anzahl der Besucher*innen pro Stunde durch die Addition von n Zufallszahlen zwischen 0 und 5 ergibt, wobei n die Anzahl der Artikel des Standes ist, die nicht `null` sind (Artikel werden später durch Verkaufen auf `null` gesetzt).
- Ein Weihnachtsartikelstand hat zwischen 1 und 20 Artikel. Sowohl die Anzahl der Artikel als auch die Artikel selbst sollen zufällig ausgewählt werden. Verwenden Sie hierfür die Methoden `Zufall.zahl(int)` und `Zufall.artikel()`, um die Anzahl, die Namen und die Preise zu bestimmen. Der Preis eines Artikels soll zwischen 0,01 Euro und 10 Euro liegen.
- Bei einem Lebensmittel ergibt sich der Preis pro 100 Gramm als Zufallszahl zwischen 0,01 Euro und 3 Euro.
- Zur Bestimmung der Süßwarenart soll die Methode `Zufall.suessware()` genutzt werden.

e) Implementieren Sie die in `Stand` als `abstract` markierte Methode `einzelkauf` in den Klassen `Weihnachtsartikelstand` und `Lebensmittelstand`. Weihnachtsartikelstände sollen alle erhältlichen Artikel auflisten und fragen, welchen Artikel die oder der Kunde*in kaufen möchte. Anschließend soll der gekaufte Artikel aus dem Sortiment gelöscht werden, indem der entsprechende Eintrag auf `null` gesetzt wird. Lebensmittelstände sollen nachfragen, wie viel Gramm die oder der Kunde*in haben möchte. Am Ende soll der Gesamtpreis zurückgegeben werden, den die oder der Kunde*in zahlen muss. Verwenden Sie hierbei die Methoden `SimpleIO.getInt(String)` und `SimpleIO.getBoolean(String)` zur Interaktion mit der oder dem Nutzer*in. Sie dürfen weitere private Hilfsmethoden anlegen, um den Code besser lesbar zu gestalten.

f) Implementieren Sie für verschiebbare Stände eine Methode `verschiebe(int standID)`, die die Anzahl der Besucher*innen pro Stunde neu berechnet und anschließend eine Meldung ausgibt, dass Stand `standID` verschoben wurde, zusammen mit der Information, von wie vielen Passanten*innen dieser Stand nun stündlich besucht wird.

Falls ein `Weihnachtsartikelstand` verschoben wird, so besteht die Möglichkeit, dass einer der Artikel beim Verschieben vom Stand fällt und kaputt geht. Der entsprechende Artikel soll aus dem Sortiment entfernt werden und es soll eine `SchadensfallException` geworfen werden, welche eine Nachricht enthält, die den Schadensfall beschreibt. Wählen Sie dazu beim Verschieben eines `Weihnachtsartikelstand`s zufällig einen Index aus dem `artikel`-Array aus. Ist dieser bereits ausverkauft (`null`), so passiert nichts. Ansonsten fällt dieser Artikel beim Verschieben vom Stand.

Legen Sie dazu die Klasse `SchadensfallException` an, welche von `Exception` erbt und einen Konstruktor enthält, der eine Fehlermeldung als `String` erhält und an den `super`-Konstruktor weitergibt. Fügen Sie außerdem eine entsprechende `throws`-Klausel für die `verschiebe`-Methode hinzu (auch am Interface `Verschiebbar`).

Sie dürfen weitere private Hilfsmethoden anlegen, um den Code besser lesbar zu gestalten.

g) Fügen Sie der Klasse `Weihnachtsmarkt` eine `main`-Methode hinzu. In dieser soll ein neuer Weihnachtsmarkt mit 5 Ständen erstellt werden und anschließend dessen `run`-Methode aufgerufen werden.

In der `run`-Methode beginnt die erste Runde, in der alle Stände des Weihnachtsmarktes aufgelistet werden und die oder der Nutzer*in gefragt wird, welchen Stand sie oder er besuchen möchte. An dem ausgewählten Stand soll die oder der Kunde*in solange Einzelkäufe tätigen können, bis sie oder er mit dem Einkauf fertig ist und den Stand verlässt. Am Ende soll der Gesamtpreis genannt werden, den die oder der Kunde*in zahlen muss.

Anschließend sollen alle verschiebbaren Stände, die von weniger als 30 Passanten*innen pro Stunde besucht werden, verschoben werden. Zum Ende einer Runde wird die oder der Nutzer*in gefragt, ob sie oder er den Weihnachtsmarkt verlassen möchte. Falls dies verneint wird, soll die nächste Runde beginnen.

Verwenden Sie die Methoden `SimpleIO.getInt(String)` und `SimpleIO.getBoolean(String)` zur Interaktion mit der oder dem Nutzer*in.

Sie dürfen weitere private Hilfsmethoden anlegen, um den Code besser lesbar zu gestalten.

Eine Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Lebensmittelstand fuer Flammkuchen:
Preis pro 100g: 0.91 Euro
Verkaeufer*in: Sarah
Besucher*innen pro Stunde: 21

1: Weihnachtsartikelstand:
Verkaeufer*in: Felix
Besucher*innen pro Stunde: 9

2: Lebensmittelstand fuer Suessware (Zuckerstange):
Preis pro 100g: 2.05 Euro
Verkaeufer*in: Mattis
Besucher*innen pro Stunde: 9

3: Weihnachtsartikelstand:
Verkaeufer*in: Per
Besucher*innen pro Stunde: 27

4: Lebensmittelstand fuer Suessware (Waffeln):
Preis pro 100g: 2.27 Euro
Verkaeufer*in: Fynn
Besucher*innen pro Stunde: 50

Welchen Stand moechten Sie besuchen?
0
Guten Tag!
Wie viel Gramm moechten Sie?
200
200 Gramm fuer Sie. Lassen Sie es sich schmecken!
Darf es sonst noch etwas sein?
false
1.82 Euro, bitte.
Stand 1 wurde verschoben und wird jetzt von 9 Passanten*innen pro Stunde besucht.
Dabei ist Artikel 1: Holzkrippe (3.7 Euro) leider vom Stand gefallen und kaputt gegangen.
Stand 2 wurde verschoben und wird jetzt von 26 Passanten*innen pro Stunde besucht.
Stand 3 wurde verschoben und wird jetzt von 33 Passanten*innen pro Stunde besucht.
Dabei ist Artikel 8: Tasse (9.1 Euro) leider vom Stand gefallen und kaputt gegangen.
Moechten Sie den Weihnachtsmarkt verlassen?
false
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Lebensmittelstand fuer Flammkuchen:
Preis pro 100g: 0.91 Euro
Verkaeufer*in: Sarah
Besucher*innen pro Stunde: 21

1: Weihnachtsartikelstand:
Verkaeufer*in: Felix
Besucher*innen pro Stunde: 9
```

```
2: Lebensmittelstand fuer Suessware (Zuckerstange):
Preis pro 100g: 2.05 Euro
Verkaeufer*in: Mattis
Besucher*innen pro Stunde: 26

3: Weihnachtsartikelstand:
Verkaeufer*in: Per
Besucher*innen pro Stunde: 33

4: Lebensmittelstand fuer Suessware (Waffeln):
Preis pro 100g: 2.27 Euro
Verkaeufer*in: Fynn
Besucher*innen pro Stunde: 50

Welchen Stand moechten Sie besuchen?
1
Guten Tag!
Unsere Artikel sind:
0: Rucksack (7.01 Euro)
1: ausverkauft
2: Kette (9.18 Euro)
Welchen Artikel moechten Sie kaufen?
0
Rucksack wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
false
7.01 Euro, bitte.
Stand 1 wurde verschoben und wird jetzt von 2 Passanten*innen pro Stunde besucht.
Stand 2 wurde verschoben und wird jetzt von 12 Passanten*innen pro Stunde besucht.
Moechten Sie den Weihnachtsmarkt verlassen?
false
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Lebensmittelstand fuer Flammkuchen:
Preis pro 100g: 0.91 Euro
Verkaeufer*in: Sarah
Besucher*innen pro Stunde: 21

1: Weihnachtsartikelstand:
Verkaeufer*in: Felix
Besucher*innen pro Stunde: 2

2: Lebensmittelstand fuer Suessware (Zuckerstange):
Preis pro 100g: 2.05 Euro
Verkaeufer*in: Mattis
Besucher*innen pro Stunde: 12

3: Weihnachtsartikelstand:
Verkaeufer*in: Per
Besucher*innen pro Stunde: 33

4: Lebensmittelstand fuer Suessware (Waffeln):
Preis pro 100g: 2.27 Euro
Verkaeufer*in: Fynn
Besucher*innen pro Stunde: 50

Welchen Stand moechten Sie besuchen?
2
Guten Tag!
Wie viel Gramm moechten Sie?
20
```

```
20 Gramm fuer Sie. Lassen Sie es sich schmecken!
Darf es sonst noch etwas sein?
true
Wie viel Gramm moechten Sie?
30
30 Gramm fuer Sie. Lassen Sie es sich schmecken!
Darf es sonst noch etwas sein?
false
1.025 Euro, bitte.
Stand 1 wurde verschoben und wird jetzt von 2 Passanten*innen pro Stunde besucht.
Stand 2 wurde verschoben und wird jetzt von 2 Passanten*innen pro Stunde besucht.
Moechten Sie den Weihnachtsmarkt verlassen?
false
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Lebensmittelstand fuer Flammkuchen:
Preis pro 100g: 0.91 Euro
Verkaeufer*in: Sarah
Besucher*innen pro Stunde: 21

1: Weihnachtsartikelstand:
Verkaeufer*in: Felix
Besucher*innen pro Stunde: 2

2: Lebensmittelstand fuer Suessware (Zuckerstange):
Preis pro 100g: 2.05 Euro
Verkaeufer*in: Mattis
Besucher*innen pro Stunde: 2

3: Weihnachtsartikelstand:
Verkaeufer*in: Per
Besucher*innen pro Stunde: 33

4: Lebensmittelstand fuer Suessware (Waffeln):
Preis pro 100g: 2.27 Euro
Verkaeufer*in: Fynn
Besucher*innen pro Stunde: 50

Welchen Stand moechten Sie besuchen?
3
Guten Tag!
Unsere Artikel sind:
0: Armband (4.87 Euro)
1: Rucksack (6.45 Euro)
2: Tasse (7.48 Euro)
3: Teelichtkarussel (8.85 Euro)
4: Uhr (4.03 Euro)
5: Kette (5.12 Euro)
6: Fensterbild (2.94 Euro)
7: Stofftier (0.88 Euro)
8: ausverkauft
Welchen Artikel moechten Sie kaufen?
2
Tasse wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
true
Unsere Artikel sind:
0: Armband (4.87 Euro)
1: Rucksack (6.45 Euro)
2: ausverkauft
3: Teelichtkarussel (8.85 Euro)
```

```
4: Uhr (4.03 Euro)
5: Kette (5.12 Euro)
6: Fensterbild (2.94 Euro)
7: Stofftier (0.88 Euro)
8: ausverkauft
Welchen Artikel moechten Sie kaufen?
4
Uhr wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
false
11.510000000000002 Euro, bitte.
Stand 1 wurde verschoben und wird jetzt von 3 Passanten*innen pro Stunde besucht.
Dabei ist Artikel 2: Kette (9.18 Euro) leider vom Stand gefallen und kaputt gegangen.
Stand 2 wurde verschoben und wird jetzt von 90 Passanten*innen pro Stunde besucht.
Moechten Sie den Weihnachtsmarkt verlassen?
true
```

Hinweise:

- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung und verwenden Sie Implementierungen in Oberklassen bzw. Interfaces soweit möglich.
- Vermeiden Sie betriebssystemspezifische Zeilenseparatoren wie `\n` bzw. `\r\n` in Strings. Verwenden Sie stattdessen `System.lineSeparator()`.