Prof. Dr. J. Giesl
S. Dollase, M. Hark, D. Cloerkes

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 **Montag, den 16.12.2019, um 12:00 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 abgeben.
- In einigen Aufgaben müssen Sie in Java programmieren und `.java`-Dateien anlegen. **Drucken** Sie diese aus **und** laden Sie sie fristgerecht im `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)" hoch. Stellen Sie sicher, dass Ihr Programm von `javac` **akzeptiert** wird, ansonsten werden keine Punkte vergeben.

  Es ist ausreichend, wenn eine Person Ihrer Abgabegruppe den Programmcode im Lernraum hochlädt. Erstellen Sie dazu ein ZIP-Archiv, welches alle `.java`-Dateien beinhaltet und laden Sie dieses im Lernraum hoch. Schreiben Sie alle Matrikelnummern der Abgabegruppe als Kommentar in Ihren Programmcode. Schreiben Sie außerdem auf Ihre schriftliche Abgabe die Matrikelnummer derjenigen Person, deren Programmcode im Lernraum bewertet werden soll.
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden. Klicken Sie dazu im `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)" rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Tutoraufgabe 1 (Programmieren in Klassenhierarchien):

In dieser Aufgabe soll ein Weihnachtsmarkt modelliert werden. Verwenden Sie hierfür die Hilfsklassen `Zufall` und `SimpleIO`, die beide im Lernraum zu finden sind.

- Ein Weihnachtsmarkt besteht aus verschiedenen Ständen.
- Ein Stand kann entweder ein Weihnachtsartikelstand oder ein Lebensmittelstand sein. Jeder Stand hat einen Verkäufer, dessen Name von Interesse ist, und eine Anzahl von Besuchern pro Stunde. Hierfür existiert sowohl ein Attribut `besucherProStunde` als auch eine Methode `berechneBesucherProStunde()`, um diese Anzahl neu zu berechnen und zurückzugeben.
- Ein Weihnachtsartikelstand hat eine Reihe an Artikeln.
- Ein Artikel hat einen Namen und einen Preis (centgenau in Euro).
- Ein Lebensmittelstand kann zum Beispiel ein Süßwarenstand, ein Glühweinstand oder ein Flammkuchenstand sein. Jeder Lebensmittelstand hat einen festen Preis (centgenau in Euro) pro 100 Gramm verkaufter Ware.
- Ein Süßwarenstand verkauft eine bestimmte Art Süßwaren.
- Im Gegensatz zu Glühwein- und 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 erhöht werden soll.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für einen Weihnachtsmarkt. 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 gleiche Notation wie in der Hausaufgabe (Aufgabe 2).

b) Implementieren Sie die Klassen entsprechend Ihrer Klassenhierarchie in einer geeigneten Struktur von Paketen und Modulen. Nutzen Sie dafür die Verzeichnisstruktur, die durch `code-weihnachtsmarkt.zip` (im Lernraum) vorgegeben ist, sodass sich Ihr Programm mit dem Befehl aus der Datei `code/compile.sh` compilieren und mit dem Befehl aus der Datei `code/run.sh` ausführen lässt,wenn man sie im Verzeichnis `code` ausführt. Hierbei soll es genau zwei Module geben: Das Modul `weihnachtsmaerkte` umfasst die Pakete und Klassen, die den Weihnachtsmarkt modellieren. Im Paket `weihnachtsmarkt.artikel` sollen alle Klassen und Interfaces liegen, welche mit Artikeln assoziiert werden könne. Im Paket `weihnachtsmarkt.staende` sollen alle Klassen und Interfaces liegen, welche mit Ständen assoziiert werden können. Das Modul `util` umfasst die Klasse `Zufall` (im Paket `zufall`) und die Klasse `SimpleIO` (im Paket `io`). Fügen Sie jeder Klasse – falls notwendig – Getter und Setter sowie eine geeignete `toString`-Methode hinzu.

Hinweise:

- Fügen Sie der Moduldefinition des Moduls `util` die Zeile `requires java.desktop;` hinzu. Dies ist notwendig, damit `SimpleIO` Zugriff auf die Java-Schnittstellen hat, welche nötig sind, um Message-Boxen anzuzeigen.

c) Fügen Sie jeder Ihrer Klassen einen Konstruktor hinzu, der ein Objekt dieser Klasse mithilfe von `Zufall.java` nach folgendem Schema erzeugt:

- 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 25% Weihnachtsartikelstände, Süßwarenstände, Glühweinstände und Flammkuchenstände gibt. Hierbei gibt ein Aufruf `Zufall.zahl(i)` (für `i` größer 0) eine zufällige Zahl zwischen 0 und `i` − 1 zurück.
- Der Name des Verkäufers eines Stands wird mit der statischen Methode `Zufall.name()` festgelegt.
- Die Anzahl der Besucher 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 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 Lebensmittelstand 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.

d) Fügen Sie der Klasse `Stand` eine Methode `verkaufe()` hinzu, die einen Kunden begrüßt und so lange Waren verkauft, bis der Kunde mit seinem Einkauf fertig ist. Weihnachtsartikelstände sollen dazu alle erhältlichen Artikel auflisten und fragen, welchen Artikel der Kunde 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 der Kunde haben möchte. Am Ende soll der Gesamtpreis genannt werden, den der Kunde zahlen muss. Fügen Sie dazu in der Klasse `Stand` eine abstrakte Methode `einzelverkauf()` hinzu, welche den Gesamtpreis zurückgibt. Diese Methode soll in den Unterklassen entsprechend implementiert werden. Verwenden Sie hierbei die Methoden `SimpleIO.getInt(String)` und `SimpleIO.getBoolean(String)` zur Interaktion mit dem Nutzer.

e) Implementieren Sie für verschiebbare Stände eine Methode `verschiebe(int i)`, die die Anzahl der Besucher pro Stunde neu berechnet und anschließend eine Meldung ausgibt, dass Stand `i` verschoben

wurde, zusammen mit der Information, von wie vielen Passanten 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. Dies soll entsprechend ausgegeben werden und der entsprechende Artikel aus dem Sortiment entfernt werden. Wählen Sie dazu beim Verschieben eines `Weihnachtsartikelstand`s zufällig einen Index aus dem `artikel`-Array aus. Ist dieser bereits ausverkauft (`null`), so passiert nicht. Ansonsten fällt dieser Artikel beim Verschieben vom Stand.

f) Fügen Sie der Klasse `Weihnachtsmarkt` eine `main`-Methode hinzu. In dieser soll ein neuer Weihnachtsmarkt mit 5 Ständen erstellt werden. Dann beginnt die erste Runde, in der alle Stände des Weihnachtsmarktes aufgelistet werden und der Nutzer gefragt wird, welchen Stand er besuchen möchte. Für den ausgewählten Stand soll ein Verkauf abgewickelt werden. Anschließend sollen alle verschiebbaren Stände, die von weniger als 30 Passanten pro Stunde besucht werden, verschoben werden. Zum Ende einer Runde wird der Nutzer gefragt, ob 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 dem Nutzer.

Eine Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Suesswarenstand (Zuckerstange):
Preis pro 100g: 1.87 Euro
Verkaeufer: Lars
Besucher pro Stunde: 39

1: Flammkuchenstand:
Preis pro 100g: 2.19 Euro
Verkaeufer: Henrik
Besucher pro Stunde: 1

2: Weihnachtsartikelstand:
Verkaeufer: Jonah
Besucher pro Stunde: 15

3: Weihnachtsartikelstand:
Verkaeufer: Carolin
Besucher pro Stunde: 34

4: Gluehweinstand:
Preis pro 100g: 0.47 Euro
Verkaeufer: Martin
Besucher pro Stunde: 99

Welchen Stand moechten Sie besuchen?
0
Guten Tag!
Wie viel Gramm moechten Sie?
150
150 Gramm fuer Sie. Lassen Sie es sich schmecken!
Darf es sonst noch etwas sein?
false
2.805 Euro, bitte.
Stand 2 wurde verschoben und wird jetzt von 19 Passanten pro Stunde besucht.
Dabei ist Artikel 2: Kuscheldecke (0.25 Euro) leider vom Stand gefallen und kaputt gegangen.
Moechten Sie den Weihnachtsmarkt verlassen?
false
```

```
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Suesswarenstand (Zuckerstange):
Preis pro 100g: 1.87 Euro
Verkaeufer: Lars
Besucher pro Stunde: 39

1: Flammkuchenstand:
Preis pro 100g: 2.19 Euro
Verkaeufer: Henrik
Besucher pro Stunde: 1

2: Weihnachtsartikelstand:
Verkaeufer: Jonah
Besucher pro Stunde: 19

3: Weihnachtsartikelstand:
Verkaeufer: Carolin
Besucher pro Stunde: 34

4: Gluehweinstand:
Preis pro 100g: 0.47 Euro
Verkaeufer: Martin
Besucher pro Stunde: 99

Welchen Stand moechten Sie besuchen?
3
Guten Tag!
Unsere Artikel sind:
0: Fensterbild (5.9 Euro)
1: Windlicht (2.32 Euro)
2: Buch (0.54 Euro)
3: Weihnachtspyramide (5.79 Euro)
4: Rucksack (7.8 Euro)
5: Buch (6.12 Euro)
6: Weihnachtsstern (8.78 Euro)
7: Stofftier (6.15 Euro)
8: Holzpuzzle (1.2 Euro)
9: Stofftier (4.43 Euro)
Welchen Artikel moechten Sie kaufen?
2
Buch wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
true
Unsere Artikel sind:
0: Fensterbild (5.9 Euro)
1: Windlicht (2.32 Euro)
2: ausverkauft
3: Weihnachtspyramide (5.79 Euro)
4: Rucksack (7.8 Euro)
5: Buch (6.12 Euro)
6: Weihnachtsstern (8.78 Euro)
7: Stofftier (6.15 Euro)
8: Holzpuzzle (1.2 Euro)
9: Stofftier (4.43 Euro)
Welchen Artikel moechten Sie kaufen?
```

```
8
Holzpuzzle wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
false
1.74 Euro, bitte.
Stand 2 wurde verschoben und wird jetzt von 21 Passanten pro Stunde besucht.
Dabei ist Artikel 4: Buch (1.14 Euro) leider vom Stand gefallen und kaputt gegangen.
Moechten Sie den Weihnachtsmarkt verlassen?
true
```

Hinweise:

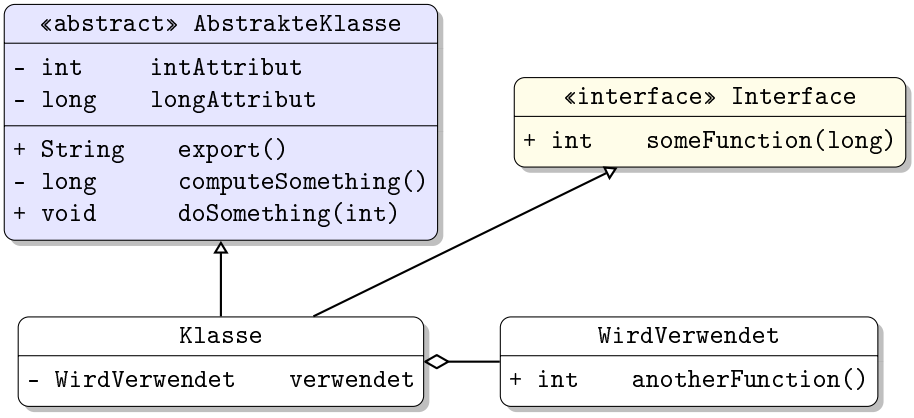
- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung und verwenden Sie Implementierungen in Oberklassen bzw. Interfaces soweit möglich.
- Vermeiden Sie betriebssystemspezifische Zeilenseparatoren wie `\n` bzw. `\r\n` in Strings. Verwenden Sie stattdessen `System.lineSeparator()`.

Aufgabe 2 (Entwurf einer Klassenhierarchie): (6 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 $B \longrightarrow A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist) und $A \diamond\!\!- B$, 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 3 (Programmieren in Klassenhierarchien): (4+4+3+3+10 = 24 Punkte)

In dieser Aufgabe soll eine grob vereinfachte Version der Prüfungsordnung für den Master Informatik modelliert werden, welche genutzt werden kann, um einen individuellen Studienplan (`Masterplan`) für einen Studierenden im Master Informatik zu erstellen. Bitte beachten Sie, dass die hier beschriebene Modellierung zwar einen guten Eindruck von der Prüfungsordnung für den Master Informatik vermittelt, ihr jedoch nicht vollständig entspricht.
Nutzen Sie für diese Aufgabe die Verzeichnisstruktur und den Programmcode, der durch `code-masterplan.zip` (im Lernraum) vorgegeben ist.
Um einen Masterplan zu erstellen, soll eine Java-Implementierung erstellt werden. Sie dient dazu, Wörter der Sprache abzubilden, die durch die folgenden Grammatikregeln in EBNF gegeben ist:

```
MasterplanBuilder = "BeginSemester" SemesterBuilder
                  | "ValidateAndCreate"

SemesterBuilder = "Anwendungsfach" CreditPoints Title SemesterBuilder
                | "Masterarbeit" Title SemesterBuilder
                | "Praktikum" Title SemesterBuilder
                | "Schwerpunktkolloquium" Title SemesterBuilder
                | "Seminar" Title Bereich SemesterBuilder
                | "Wahlpflichtvorlesung" CreditPoints Title Bereich SemesterBuilder
                | "EndSemester" MasterplanBuilder

CreditPoints = ein int-Wert
Title = ein String-Wert
Bereich = Bereich.THEORETISCHE_INFORMATIK
        | Bereich.SOFTWARE_UND_KOMMUNIKATION
        | Bereich.DATEN_UND_INFORMATIONSMANAGEMENT
        | Bereich.ANGEWANDTE_INFORMATIK
```

Ein Wort, welches aus dem Nicht-Terminal-Symbol `MasterplanBuilder` hergeleitet werden kann, sieht z.B. wie folgt aus:

```
  BeginSemester
    Wahlpflichtvorlesung 6 "Funktionale Programmierung" Bereich.THEORETISCHE_INFORMATIK
    Wahlpflichtvorlesung 6 "Software-Projektmanagement" Bereich.SOFTWARE_UND_KOMMUNIKATION
    Wahlpflichtvorlesung 6 "IT-Sicherheit" Bereich.DATEN_UND_INFORMATIONSMANAGEMENT
    Wahlpflichtvorlesung 6 "Infinite Computations and Games" Bereich.THEORETISCHE_INFORMATIK
    Anwendungsfach 6 "Investition und Finanzierung (BWL)"
  EndSemester
  BeginSemester
    Schwerpunktkolloquium "Programmverifikation"
    Masterarbeit "Modular Heap Shape Analysis for Java Programs"
  EndSemester
ValidateAndCreate
```

Die Semantik davon ist ein Masterplan mit zwei Semestern. Im ersten Semester werden vier Wahlpflichtvorlesungen sowie ein Anwendungsfach belegt. Im zweiten Semester wird ein Schwerpunktkolloquium und eine Masterarbeit absolviert.
Dieses soll mit unserer Java-Implementierung durch folgenden Aufruf ausgedrückt werden können:

```
Masterplan masterplan = Masterplan.newBuilder()
    .beginSemester()
      .wahlpflichtvorlesung(6, "Funktionale Programmierung", Bereich.THEORETISCHE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Software-Projektmanagement", Bereich.SOFTWARE_UND_KOMMUNIKATION)
      .wahlpflichtvorlesung(6, "IT-Sicherheit", Bereich.DATEN_UND_INFORMATIONSMANAGEMENT)
      .wahlpflichtvorlesung(6, "Infinite Computations and Games", Bereich.THEORETISCHE_INFORMATIK)
      .anwendungsfach(6, "Investition und Finanzierung (BWL)")
    .endSemester()
```

```
    .beginSemester()
      .schwerpunktkolloquium("Programmverifikation")
      .masterarbeit("Modular Heap Shape Analysis for Java Programs")
    .endSemester()
  .validateAndCreate();
```

Dazu haben wir die obigen Grammatikregeln für die Nicht-Terminal-Symbole `MasterplanBuilder` und `SemesterBuilder` in Form von Interfaces kodiert (siehe die Interfaces `MasterplanBuilder` und `SemesterBuilder`). Außerdem liegt das Nicht-Terminal-Symbol `Bereich` als Enum `Bereich` vor.[1]
Die Auswertung des obigen Java-Ausdrucks soll nun dazu führen, dass eine `InvalidMasterplanException` geworfen wird, da er sich zwar an die Ableitungsregeln der Grammatik hält, jedoch keinen gültigen Masterplan darstellt, denn es werden z.B. keine 120 Credit Points erreicht. Einen solchen Masterplan nennen wir "nicht valide". Was genau einen validen Masterplan ausmacht, ist in Aufgabenteil e) beschrieben.
Ein valider Masterplan würde z.B. wie folgt aussehen (siehe Klasse `Launcher`):

```
Masterplan masterplan = Masterplan.newBuilder()
    .beginSemester()
      .wahlpflichtvorlesung(6, "Funktionale Programmierung", Bereich.THEORETISCHE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Compilerbau", Bereich.THEORETISCHE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Statische Programmanalyse", Bereich.THEORETISCHE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Infinite Computations and Games", Bereich.THEORETISCHE_INFORMATIK)
      .anwendungsfach(6, "Investition und Finanzierung (BWL)")
    .endSemester()
    .beginSemester()
      .seminar("Secure Crypto Protocol System", Bereich.THEORETISCHE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Software-Projektmanagement", Bereich.SOFTWARE_UND_KOMMUNIKATION)
      .wahlpflichtvorlesung(6, "IT-Sicherheit", Bereich.DATEN_UND_INFORMATIONSMANAGEMENT)
      .wahlpflichtvorlesung(6, "Objektorientierte Softwarekonstruktion", Bereich.SOFTWARE_UND_KOMMUNIKATION)
      .anwendungsfach(6, "Methoden und Anwendungen der Optimierung (BWL)")
    .endSemester()
    .beginSemester()
      .praktikum("Secure Crypto Protocol System")
      .wahlpflichtvorlesung(6, "Machine Learning", Bereich.ANGEWANDTE_INFORMATIK)
      .wahlpflichtvorlesung(6, "Communication Systems Engineering", Bereich.SOFTWARE_UND_KOMMUNIKATION)
      .wahlpflichtvorlesung(6, "Introduction to Data Science", Bereich.DATEN_UND_INFORMATIONSMANAGEMENT)
      .anwendungsfach(6, "Advanced Operations Research (BWL)")
    .endSemester()
    .beginSemester()
      .schwerpunktkolloquium("Programmverifikation")
      .masterarbeit("Modular Heap Shape Analysis for Java Programs")
    .endSemester()
  .validateAndCreate();
```

Die Klasse `Launcher` würde hier folgende Ausgabe produzieren:

```
Semester 1
6 CP Funktionale Programmierung (Wahlpflichtvorlesung im Bereich Theoretische Informatik)
6 CP Compilerbau (Wahlpflichtvorlesung im Bereich Theoretische Informatik)
6 CP Statische Programmanalyse (Wahlpflichtvorlesung im Bereich Theoretische Informatik)
6 CP Infinite Computations and Games (Wahlpflichtvorlesung im Bereich Theoretische Informatik)
6 CP Investition und Finanzierung (BWL) (Anwendungsfach)

Semester 2
4 CP Secure Crypto Protocol System (Seminar im Bereich Theoretische Informatik)
6 CP Software-Projektmanagement (Wahlpflichtvorlesung im Bereich Software und Kommunikation)
6 CP IT-Sicherheit (Wahlpflichtvorlesung im Bereich Daten- und Informationsmanagement)
6 CP Objektorientierte Softwarekonstruktion (Wahlpflichtvorlesung im Bereich Software und Kommunikation)
6 CP Methoden und Anwendungen der Optimierung (BWL) (Anwendungsfach)

Semester 3
7 CP Secure Crypto Protocol System (Praktikum)
6 CP Machine Learning (Wahlpflichtvorlesung im Bereich Angewandte Informatik)
6 CP Communication Systems Engineering (Wahlpflichtvorlesung im Bereich Software und Kommunikation)
6 CP Introduction to Data Science (Wahlpflichtvorlesung im Bereich Daten- und Informationsmanagement)
6 CP Advanced Operations Research (BWL) (Anwendungsfach)

Semester 4
```

[1] Es gibt verschiedene Begriffe für diese Art der Programmierung, wie z.B. *Fluent Interface*, *Method Chaining* oder *Internal Domain Specific Language (Internal DSL)*.

```
3 CP Programmverifikation (Schwerpunktkolloquium)
30 CP Modular Heap Shape Analysis for Java Programs (Masterarbeit)
```

a) Implementieren Sie alle Klassen und Interfaces in einer geeigneten Struktur von Paketen und Modulen. Nutzen Sie dafür die Verzeichnisstruktur und den Programmcode, der im Lernraum vorgegeben ist, sodass sich Ihr Programm mit dem Befehl aus der Datei `code/compile.sh` compilieren und mit dem Befehl aus der Datei `code/run.sh` ausführen lässt, wenn man sie im Verzeichnis `code` ausführt. Hierbei soll es genau zwei Module geben: Das Modul `masterplan` umfasst die Pakete, Klassen und Interfaces, die den Masterplan modellieren, und das Modul `launcher` umfasst die Klasse `Launcher` im Paket `launcher`. Das Modul `masterplan` enthält drei Pakete. Das Paket `masterplan` ist von anderen Modulen zugreifbar. Das Paket `masterplan.impl` und das Paket `masterplan.test` ist nicht von anderen Modulen zugreifbar. Stellen Sie dies über entsprechende `module-info.java`-Dateien sicher und achten Sie darauf, die neu erstellten Klassen und Interfaces im korrekten Paket zu erstellen.

Das Paket `masterplan.test` enthält ausschließlich Code um Ihre Implementierung zu testen. Um die Tests aus der Klasse `masterplan.test.MasterplanTest` ausführen zu können, müssen Sie zunächst zwei Libraries herunterladen:

- `https://search.maven.org/remotecontent?filepath=junit/junit/4.13-rc-2/junit-4.13-rc-2.jar`
- `https://search.maven.org/remotecontent?filepath=org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.jar`

Legen Sie beide Libraries im `code`-Ordner ab. Es sollten nun die beiden Dateien `code/junit-4.13-rc-2.jar` und `code/hamcrest-core-1.3.jar` vorhanden sein. Führen Sie zuerst den Befehl aus der Datei `code/compile-tests.sh` im Ordner `code` aus. Führen Sie anschließend den Befehl aus der Datei `code/run-tests.sh` im Ordner `code` aus.

b) Implementieren Sie das Interface `Lehrveranstaltung` aus dem Modul `masterplan` durch die abstrakte Klasse `LehrveranstaltungBase` im Modul `masterplan`. Diese Klasse speichert die Credit Points sowie eine Beschreibung der Lehrveranstaltung in je einem Attribut. Sie implementiert die beiden Methoden des Interfaces, indem sie die entsprechenden Attributwerte zurückgibt. Außerdem enthält sie einen Konstruktor, welcher drei Parameter erhält und die beiden Attribute wie folgt zuweist. Der erste Parameter (`creditPoints`) wird einfach dem entsprechenden Attribut zugewiesen. Die anderen beiden Parameter sind vom Typ `String` und werden folgendermaßen zu einer Beschreibung zusammengesetzt. Wenn der zweite Parameter (`title`) den Wert `"Funktionale Programmierung"` hat und der dritte Parameter (`type`) den Wert `"Vorlesung"` hat, dann wird dem Attribut folgender Wert zugewiesen: `"Funktionale Programmierung (Vorlesung)"`.

Implementieren Sie eine weitere abstrakte Klasse `LehrveranstaltungMitBereichszuordnung`, welche die Klasse `LehrveranstaltungBase` erweitert. Der Konstruktor dieser Klasse erhält vier Parameter, wobei die ersten beiden (`creditPoints` und `title`) einfach an den Super-Konstruktor weitergereicht werden. Auch der dritte Parameter (`type`) wird einfach durchgereicht, jedoch wird hier zusätzlich der Text `" im Bereich Theoretische Informatik"` angehängt, falls der vierte Parameter (`bereich` vom Typ `Bereich`) den Wert `Bereich.THEORETISCHE_INFORMATIK` hat. Analog dazu geht man vor, wenn der Parameter `bereich` einen anderen Wert hat. Außerdem soll der Konstruktor den vierten Parameter in einem Attribut ablegen und über einen Getter abrufbar machen.

Hinweise:

- Der Bereich beschreibt, in welchem der vier Wahlpflichtbereiche eine Lehrveranstaltung angesiedelt ist. Nicht alle Lehrveranstaltungen sind einem Bereich zugeordnet. Die vier Bereiche sind über das Enum `masterplan.Bereich` im Modul `masterplan` modelliert. Zu einem gegebenen `bereich` kann eine Beschreibung mit `bereich.getDescription()` abgerufen werden.
- Beide Klassen sollen von anderen Modulen aus nicht zugreifbar sein.

c) Nun wollen wir die konkreten Lehrveranstaltungstypen je als eigene Klasse implementieren. Sorgen Sie dafür, dass jede dieser Klassen die geeignete Superklasse (`LehrveranstaltungBase` oder `LehrveranstaltungMitBereichszuordnung`) erweitert und über einen Konstruktor verfügt, welcher die notwendigen Eingabe-Parameter erhält und den entsprechende Super-Konstruktor korrekt aufruft.

- Ein `Anwendungsfach` ist **keinem** Bereich zugeordnet. Der `type` eines Anwendungsfachs ist `"Anwendungsfach"`.

- Ein `Masterarbeit` ist **keinem** Bereich zugeordnet. Der `type` einer Masterarbeit ist `"Masterarbeit"`. Der Abschluss einer Masterarbeit wird mit 30 Credit Points gewertet.
- Ein `Praktikum` ist **keinem** Bereich zugeordnet. Der `type` eines Praktikums ist `"Praktikum"`. Der Abschluss eines Praktikums wird mit 7 Credit Points gewertet.
- Ein `Schwerpunktkolloquium` ist **keinem** Bereich zugeordnet. Der `type` eines Schwerpunktkolloquiums ist `"Schwerpunktkolloquium"`. Der Abschluss eines Praktikums wird mit 3 Credit Points gewertet.
- Ein `Seminar` ist **einem** Bereich zugeordnet. Der `type` eines Seminars ist `"Seminar"`. Der Abschluss eines Seminars wird mit 4 Credit Points gewertet.
- Eine `Wahlpflichtvorlesung` ist **einem** Bereich zugeordnet. Der `type` einer Wahlpflichtvorlesung ist `"Wahlpflichtvorlesung"`.

Hinweise:

- Keine dieser Klassen soll von anderen Modulen aus zugreifbar sein.

d) Erstellen Sie die Klasse `MasterplanImpl`, welche das Interface `Masterplan` implementiert. Dazu bekommt sie ein zweidimensionales Array mit Einträgen vom Typ `Lehrveranstaltung` als Parameter an ihren Konstruktor übergeben, welcher das Array in ein Attribut schreibt. Die erste Dimension des Arrays ist der Semesterzähler. Die zweite Dimension des Arrays ist der Lehrveranstaltungszähler. Der Arrayeintrag `semesters[1][3]` soll also die vierte Lehrveranstaltung im zweiten Semester enthalten.

Die Methode `getNumberOfSemesters` soll zurückgeben, wie viele Semester in dem Array enthalten sind, das dem Konstruktor übergeben wurde.

Die Methode `getNumberOfLehrveranstaltungen` soll zurückgeben, wie viele Lehrveranstaltungen für das entsprechende Semester in dem Array enthalten sind, das dem Konstruktor übergeben wurde.

Die Methode `getLehrveranstaltung` gibt die Lehrveranstaltung am entsprechenden Index des Arrays zurück, das dem Konstruktor übergeben wurde.

Hinweise:

- Achten Sie insbesondere darauf, im Fehlerfall die richtige Exception zu werfen. Beachten Sie dazu die Javadoc-Dokumentation des Interfaces `Masterplan`. Es soll also bei negativem oder zu großem Index keine `ArrayIndexOutOfBoundsException` geworfen werden, sondern eine `IllegalArgumentException`, sodass Nutzer des Interfaces `Masterplan` nicht wissen müssen, dass die Implementierung `MasterplanImpl` intern mit Arrays arbeitet.
- Diese Klasse soll von anderen Modulen aus nicht zugreifbar sein.

e) Erstellen Sie die Klasse `BuilderImpl`, welche die beiden Interfaces `MasterplanBuilder` und `SemesterBuilder` implementiert. Mit einem Objekt dieser Klasse kann ein Masterplan erstellt werden, welcher bis zu 10 Semester und bis zu 10 Lehrveranstaltungen pro Semester enthalten kann. Es soll also ein Attribut `semesters` vom Typ `Lehrveranstaltung[][]` besitzen, welches mit einem neuen zweidimensionalen Array initialisiert wird, welches pro Dimension 10 Einträge halten kann. Außerdem besitzen diese Objekte zwei weitere `int`-Attribute (`currentSemester` und `currentLehrveranstaltung`), welche mit 0 initialisiert werden.

Die Methode `beginSemester` überprüft, ob der Wert von `currentSemester` ein gültiger Index im Array `semesters` ist. Ist dies nicht der Fall, so wird eine `InvalidMasterplanException` geworfen. Ansonsten wird `this` zurückgegeben.

Die Methode `endSemester` erhöht `currentSemester` um eins und setzt `currentLehrveranstaltung` auf 0 zurück. Außerdem gibt sie `this` zurück.

Die Methoden `anwendungsfach`, `masterarbeit`, `praktikum`, `schwerpunktkolloquium`, `seminar` und `wahlpflichtvorlesung` erstellen je im aktuellen Semester die entsprechende Lehrveranstaltung mit den entsprechenden Parametern. Dazu überprüfen Sie zunächst, ob im aktuellen Semester noch Platz für eine weitere Lehrveranstaltung ist. Ist dies nicht der Fall, so wird eine `InvalidMasterplanException` geworfen. Ansonsten erstellen Sie zunächst ein Objekt der entsprechenden Unterklasse von `Lehrveranstaltung` und schreiben es an die Stelle `semesters[currentSemester][currentLehrveranstaltung]`. Anschließend wird `currentLehrveranstaltung` um eins erhöht und `this` zurückgegeben.

Die Methode `validateAndCreate` schrumpft das `semesters` Array zunächst (siehe unten) und erstellt anschließend ein `Masterplan`-Objekt mit dem geschrumpften Array und gibt dieses zurück. Zuvor überprüft sie jedoch, ob der konfigurierte Masterplan valide ist. Ist dies nicht der Fall, so wird stattdessen eine `InvalidMasterplanException` geworfen.

Das Schrumpfen des Arrays funktioniert so, dass aus dem `semesters`-Array ein neues Array mit demselben Inhalt generiert wird, in dem jedoch die `null`-Einträge fehlen. Dazu muss die Größe des neuen Arrays korrekt gewählt werden. Falls etwa ein Masterplan konfiguriert wurde, welcher aus zwei Semestern besteht (d.h. alle weiteren Semester sind `null`) und im ersten Semester 3 und im zweiten Semester 4 Lehrveranstaltungen enthält (d.h. alle weiteren Lehrveranstaltungen der jeweiligen Semester sind `null`), dann soll das geschrumpfte Array die Größe 2 haben (Semesteranzahl) und im ersten Eintrag ein Array der Größe 3 enthalten (Lehrveranstaltungsanzahl im ersten Semester) und im zweiten Eintrag ein Array der Größe 4 enthalten (Lehrveranstaltungsanzahl im zweiten Semester). Gehen Sie hierbei stets davon aus, dass in einem Array nach einem `null`-Eintrag keine weiteren nicht-`null`-Einträge kommen.

Damit ein Masterplan valide ist müssen folgende Bedingungen erfüllt sein.

- Es müssen mindestens 12 Credit Points aus dem Bereich der Theoretischen Informatik belegt worden sein.
- Pro Bereich dürfen höchstens 35 Credit Points belegt worden sein.
- Es müssen Lehrveranstaltungen aus mindestens drei verschiedenen Bereichen belegt worden sein.
- Für das Anwendungsfach müssen genau 18 Credit Points belegt worden sein.
- Es wurde je genau ein Seminar, ein Praktikum, ein Schwerpunktkolloquium sowie eine Masterarbeit belegt.
- Insgesamt wurden mindestens 120 Credit Points belegt.

Hinweise:

- Achten Sie darauf, eine aussagekräftige Fehlerbeschreibung zu hinterlegen, falls Sie einen Fehler werfen.
- Diese Klasse soll von anderen Modulen aus nicht zugreifbar sein.
- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung.

Aufgabe 4 (Deck 7): (Codescape)

Schließen Sie das Spiel Codescape ab, indem Sie den letzten Raum auf Deck 7 auf eine der drei möglichen Arten lösen. Genießen Sie anschließend das Outro. Dieses Deck enthält keine für die Zulassung relevanten Missionen.

Hinweise:

- Es gibt 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.