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 12.12.2017 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 **Vorlesung** abgeben. **Die zugehörige Globalübung findet erst am Freitag, den 15.12.2017 um 8:30 Uhr statt.**
- 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 12.12.2017 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.
- 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 (Überschreiben, Überladen und Verdecken):

Betrachten Sie die folgenden Klassen:

Listing 1: X.java

```
1  public class X {
2      public int a = 23;
3      public X (int a) {                          // Signatur: X(I)
4          super();
5          this.a = a;
6      }
7
8      public X (float x) {                        // Signatur: X(F)
9          this((int) (x + 1));
10     }
11
12     public void f(int  i, X o) { }              // Signatur: X.f(IX)
13     public void f(long lo, Y o) { }             // Signatur: X.f(LY)
14     public void f(long lo, X o) { }             // Signatur: X.f(LX)
15 }
```

Listing 2: Y.java

```
1  public class Y extends X {
2      public float a = 42;
3      public Y (double a) {                       // Signatur: Y(D)
4          this((float) (a - 1));
5      }
```

```
6       public Y (float a) {                          // Signatur: Y(F)
7           super(a);
8           this.a = a;
9       }
10      public void f(int i, X o) { }                 // Signatur: Y.f(IX)
11      public void f(int i, Y o) { }                 // Signatur: Y.f(IY)
12      public void f(long lo, X o) { }               // Signatur: Y.f(LX)
13  }
```

Listing 3: Z.java

```
1   public class Z {
2       public static void main (String [] args) {
3                                                     // a)
4           X xx1 = new X(42);                        // (1)
5           System.out.println("X.a: " + xx1.a);
6           X xx2 = new X(22.99f);                    // (2)
7           System.out.println("X.a: " + xx2.a);
8           X xy = new Y(7.5);                        // (3)
9           System.out.println("X.a: " + ((X) xy).a);
10          System.out.println("Y.a: " + ((Y) xy).a);
11          Y yy = new Y(7);                          // (4)
12          System.out.println("X.a: " + ((X) yy).a);
13          System.out.println("Y.a: " + ((Y) yy).a);
14                                                    // b)
15          int i = 1;
16          long lo = 2;
17          xx1.f(i, xy);                             // (1)
18          xx1.f(lo, xx1);                           // (2)
19          xx1.f(lo, yy);                            // (3)
20          yy.f(i, yy);                              // (4)
21          yy.f(i, xy);                              // (5)
22          yy.f(lo, yy);                             // (6)
23          xy.f(i, xx1);                             // (7)
24          xy.f(lo, yy);                             // (8)
25          //xy.f(i, yy);                            // (9)
26      }
27  }
```

In dieser Aufgabe sollen Sie angeben, welche Methoden- und Konstruktoraufrufe stattfinden. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Verwenden Sie zur eindeutigen Bezeichnung die Funktionssignatur, die jeweils als Kommentar hinter jeder Funktionsdefinition steht. Begründen Sie Ihre Antwort kurz.

a) Geben Sie für die mit `(1)-(4)` markierten Konstruktoraufrufe in der Klasse `Z` jeweils an, welche Konstruktoren in welcher Reihenfolge von Java aufgerufen werden. Bedenken Sie, dass die Oberklasse von `X` die Klasse `Object` ist. Erklären Sie außerdem, welche Attribute mit welchen Werten belegt werden und welche Werte durch die `println`-Anweisungen ausgegeben werden.

b) Geben Sie für die mit `(1)-(9)` markierten Aufrufe der Methode `f` in der Klasse `Z` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Aufgabe 2 (Überschreiben, Überladen und Verdecken): (4+3 = 7 Punkte)

Betrachten Sie die folgenden Klassen:

Listing 4: A.java

```
1   public class A {
2       public static int x = 0;
3       public int y = 7;
4
5       public A () {                                 // Signatur: A()
6           this(x);
7           x++;
8       }
9
10      public A (int x) {                            // Signatur: A(I)
11          x = x + 2;
12          y = y - x;
13      }
14
15      public A (double x) {                         // Signatur: A(D)
```

```
16          y += x;
17      }
18
19      public void f(int i, A o) { }                  // Signatur: A.f(IA)
20      public void f(Long lo, A o) { }                 // Signatur: A.f(LA)
21      public void f(double d, A o) { }               // Signatur: A.f(DA)
22  }
```

Listing 5: B.java

```
1   public class B extends A {
2       public float x = 1.5f;
3       public int y = 1;
4
5       public B () {                                  // Signatur: B()
6           x++;
7       }
8
9       public B (float x) {                           // Signatur: B(F)
10          super(x);
11          super.y++;
12      }
13
14      public void f(int i, B o) { }                  // Signatur: B.f(IB)
15      public void f(int i, A o) { }                  // Signatur: B.f(IA)
16      public void f(long lo, A o) { }                // Signatur: B.f(LA)
17  }
```

Listing 6: C.java

```
1   public class C {
2       public static void main (String [] args) {
3           // a)
4           A a1 = new A(5);                           // (1)
5           System.out.println (A.x);
6           System.out.println (a1.y);
7           A a2 = new A();                            // (2)
8           System.out.println (A.x);
9           System.out.println (a2.y);
10          B b = new B();                             // (3)
11          System.out.println (A.x);
12          System.out.println (b.x);
13          System.out.println (((A) b).y);
14          System.out.println (b.y);
15          A ab = new B(3);                           // (4)
16          System.out.println (ab.y);
17          System.out.println (((B) ab).y);
18          // b)
19          int i = 1;
20          long l = 2;
21          double d = 3.0;
22          a1.f(i, a1);                               // (1)
23          a1.f(l, ab);                               // (2)
24          b.f(i, (B) ab);                            // (3)
25          b.f(d, ab);                                // (4)
26          ab.f(i, a1);                               // (5)
27          ab.f(i, b);                                // (6)
28      }
29  }
```

In dieser Aufgabe sollen Sie angeben, welche Methoden- und Konstruktoraufrufe stattfinden. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Verwenden Sie zur eindeutigen Bezeichnung die Funktionssignatur, die jeweils als Kommentar hinter jeder Funktionsdefinition steht. Begründen Sie Ihre Antwort kurz.

a) Geben Sie für die mit `(1)`-`(4)` markierten Konstruktoraufrufe in der Klasse `C` jeweils an, welche Konstruktoren in welcher Reihenfolge von Java aufgerufen werden. Bedenken Sie, dass die Oberklasse von `A` die Klasse `Object` ist. Erklären Sie außerdem, welche Werte durch die `println`-Anweisungen ausgegeben werden.

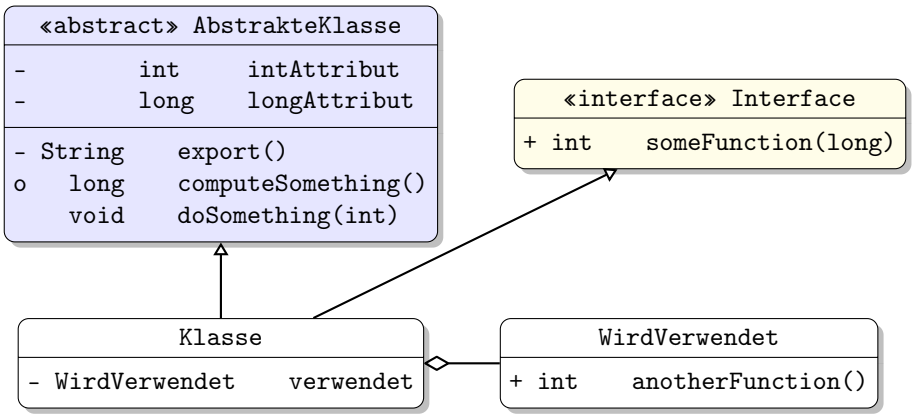
b) Geben Sie für die mit `(1)`-`(8)` markierten Aufrufe der Methode `f` in der Klasse `C` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Tutoraufgabe 3 (Entwerfen von Klassenhierarchien):

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 $A \diamond\!\!- B$, dass A den Typ B verwendet (z.B. als Typ eines Attributs oder in der Signatur einer Methode). Benutzen sie `+`, `-` und `o` um `public`, `private` und `protected` abzukürzen.
Tragen Sie keine vordefinierten Klassen (String, etc.) oder Pfeile dorthin in ihr Diagramm ein.

Tutoraufgabe 4 (Programmieren in Klassenhierarchien):

In einer kleinen Stadt kommt es gehäuft zu Diebstählen. Sperren Sie die Diebe weg! Verwenden Sie hierbei die Hilfsklasse `Zufall` (auf der Homepage), die zwei statische Methoden `int zahl(int)` und `String name()` enthält. Ein Aufruf `Zufall.zahl(i)` (für `i` größer 0) gibt eine zufällige Zahl zwischen 0 und `i` − 1 zurück. Ein Aufruf `Zufall.name()` gibt einen zufällig gewählten Namen zurück.
Implementieren Sie alle Klassen in dieser Aufgabe in einer geeigneten Struktur von Paketen und Modulen. Nutzen Sie dafür die Verzeichnisstruktur, die durch `t4.zip` (auf der Webseite) vorgegeben ist. Sorgen Sie dafür, dass sich Ihr Programm mit dem Befehl aus der Datei `t4/compile.sh` compilieren und mit dem Befehl aus der Datei `t4/run.sh` ausführen lässt.

a) Schreiben Sie die Klasse `Buerger`, welche die "normalen" Bürger der Stadt modelliert. Ein Bürger hat einen Namen (als `String` repräsentiert), der an den Konstruktor übergeben wird und bei einem Aufruf der `toString`-Methode zurückgegeben wird. Speichern Sie den Namen in einem Attribut, welches die Sichtbarkeit `private` hat. Da sich der Name eines Bürgers nicht ändern soll, definieren Sie nur eine `get`-Methode, aber keine `set`-Methode. Weiterhin hat jeder Bürger die Methode `hatDiebesgut`, die `false` zurück gibt. Ein Bürger kann eine Aktion ausführen. Dies geschieht durch die Methode `void aktion(Buerger [] einwohner)`. Bei dem Aufruf dieser Methode soll für einen normalen Bürger ausgegeben werden, dass der Bürger spazieren geht. Das Argument `einwohner` wird bei normalen Bürgern nicht benötigt, Sie können aber immer davon ausgehen, dass es keine `null`-Werte enthält.

b) Ein reicher Bürger wird in der Klasse `ReicherBuerger` repräsentiert, welche die Klasse `Buerger` erweitert. Ein reicher Bürger hat einen gewissen Reichtum in Euro, der durch ein Attribut `reichtum` vom Typ `int` dargestellt wird. Der Konstruktor eines reichen Bürgers bekommt Namen und Reichtum als Parameter übergeben und initialisiert das Objekt entsprechend. Die Aktion (implementiert in der Methode `void aktion(Buerger [] einwohner)`) eines reichen Bürgers besteht darin, mit einem (zufälligen) Teil seines Geldes (seinen letzten Euro behält der reiche Bürger allerdings selbst) Politiker zu bestechen, was als Mitteilung ausgegeben werden soll. Dadurch schrumpft sein Reichtum um den entsprechenden Betrag.

c) Ein Dieb ist ebenfalls ein Bürger und wird durch die Klasse `Dieb` repräsentiert. Ein Dieb hat ein Attribut `int diebesgut`, welches durch den Konstruktor auf 0 initialisiert wird. Somit hat der Konstruktor nur den Parameter `name`. Die Methode `hatDiebesgut` soll zurückgeben, ob das Diebesgut größer als 0 ist. Bei einem Aufruf der Methode `void aktion(Buerger [] einwohner)` hat der Dieb 5 Versuche, um Bürger zu bestehlen. In jedem Versuch soll ein Bürger aus dem Array `einwohner` zufällig ausgewählt werden. Ist es ein reicher Bürger, der mindestens einen Euro besitzt, so klaut der Dieb ihm einen zufälligen Teil seines Reichtums (aber nicht seinen letzten Euro). Hierbei wird das Attribut `reichtum` des reichen Bürgers um den Betrag verringert, durch den sich das Attribut `diebesgut` des Diebes erhöht. Trifft der Dieb bei den Versuchen auf einen Polizisten, so bricht er die Aktion ab (die restlichen Versuche verfallen).

d) Ein Gefangener ist ein Dieb, der kein Diebesgut besitzt und im Gefängnis sitzt. Schreiben Sie die Klasse `Gefangener`. Diese verfügt über einen Konstruktor, der den Namen des Gefangenen als einziges Argument entgegen nimmt. Außerdem verfügt sie über eine Methode `void aktion(Buerger[] einwohner)`, die auf der Konsole ausgibt, dass sich der Gefangene mit dem Namen, der dem Konstruktor als Argument übergeben wurde, im Gefängnis ärgert.

e) Ein Polizist ist ein Bürger, der Verbrecher jagt. Bei einem Aufruf der Methode `void aktion(Buerger [] einwohner)` sucht der Polizist bei den Bürgern im Array `einwohner` nach Diebesgut. Hierzu durchläuft er das Array von vorne nach hinten und verwendet die Methode `hatDiebesgut` der Bürger. Findet der Polizist Diebesgut, so ersetzt er den Dieb an der Stelle im Array durch einen Gefangenen gleichen Namens. Außerdem verfügt die Klasse Polizist über einen Konstruktor, der den Namen des Polizisten als einziges Argument entgegen nimmt.

f) Erstellen Sie die Klasse `Stadt`, welche das als `private` markierte Attribut `Buerger[] einwohner` besitzt. Der Konstruktor dieser Klasse bekommt das Argument `int einwohnerzahl` und legt ein Array mit entsprechend vielen Bürgern an, deren Namen (mit der Methode `Zufall.name()`) zufällig bestimmt werden sollen. Verwenden Sie die Methode `Zufall.zahl(int)` so, dass es jeweils etwa 20% Diebe, Reiche Bürger, Polizisten, Gefangene und normale Bürger gibt. Der Reichtum eines reichen Bürgers soll zwischen 0 und 999 Euro liegen. In der statischen `main`-Methode der Klasse soll eine neue Stadt mit 10 Bürgern erstellt werden. Danach wird 10 mal ein zufälliger Bürger ausgewählt und seine Methode `aktion` mit allen Bürgern der Stadt aufgerufen.

Eine Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Dieb Wibke sucht nach Diebesgut.
Dieb Wibke klaut Jannik 199 Euro.
Reicher Buerger Hanna besticht einen Politiker mit 37 Euro.
Buerger Madeleine geht spazieren.
Polizist Niels geht auf Verbrecherjagd.
Polizist Niels entlarvt Dieb Wibke.
Dieb Wibke wurde eingesperrt.
```

```
Buerger Bryan geht spazieren.
Dieb Max sucht nach Diebesgut.
Dieb Max bricht die Suche ab
Polizist Niels geht auf Verbrecherjagd.
Gefangener Dominic aergert sich im Gefaengnis.
Buerger Mohamed geht spazieren.
Reicher Buerger Jannik besticht einen Politiker mit 62 Euro.
```

Hinweise:

- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung.

Aufgabe 5 (Programmieren in Klassenhierarchien): (4+5+5+4+2+3 = 23 Punkte)

In dieser Aufgabe soll ein Weihnachtsmarkt modelliert werden. Verwenden Sie hierfür die Hilfsklassen `Zufall` und `SimpleIO`, die beide auf der Homepage 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 zu berechnen.
- 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 Tutoraufgabe 3.

b) Implementieren Sie die Klassen entsprechend Ihrer Klassenhierarchie in einer geeigneten Struktur von Paketen und Modulen. Nutzen Sie dafür die Verzeichnisstruktur, die durch `h5.zip` (auf der Webseite) vorgegeben ist, sodass sich Ihr Programm mit dem Befehl aus der Datei `h5/compile.sh` compilieren und mit dem Befehl aus der Datei `h5/run.sh` ausführen lässt. Hierbei soll es genau zwei Module geben: Das Modul `weihnachtsmaerkte` umfasst die Pakete und Klassen, die den Weihnachtsmarkt modellieren, und 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 der Klasse `SimpleIO` die Zeile `requires java.desktop;` hinzu.

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.
- 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. Verwenden Sie hierbei geeignete Hilfsmethoden und 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.

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: Gluehweinstand:
Preis pro 100g: 1.62 Euro
Verkaeufer: Janna
Besucher pro Stunde: 88

1: Flammkuchenstand:
Preis pro 100g: 2.7 Euro
Verkaeufer: Eric
Besucher pro Stunde: 64
```

```
2: Suesswarenstand (gebrannte Mandeln):
Preis pro 100g: 1.48 Euro
Verkaeufer: Judith
Besucher pro Stunde: 95

3: Flammkuchenstand:
Preis pro 100g: 0.92 Euro
Verkaeufer: Kristin
Besucher pro Stunde: 57

4: Weihnachtsartikelstand:
Verkaeufer: Sandro
Besucher pro Stunde: 15

Welchen Stand moechten Sie besuchen?
2
Guten Tag!
Wie viel Gramm moechten Sie?
150
150 Gramm fuer Sie. Lassen Sie es sich schmecken!
Darf es sonst noch etwas sein?
false
2.22 Euro, bitte.
Stand 4 wurde verschoben und wird jetzt von 27 Passanten pro Stunde besucht.
Moechten Sie den Weihnachtsmarkt verlassen?
false
Der Weihnachtsmarkt besteht aus folgenden Staenden:

0: Gluehweinstand:
Preis pro 100g: 1.62 Euro
Verkaeufer: Janna
Besucher pro Stunde: 88

1: Flammkuchenstand:
Preis pro 100g: 2.7 Euro
Verkaeufer: Eric
Besucher pro Stunde: 64

2: Suesswarenstand (gebrannte Mandeln):
Preis pro 100g: 1.48 Euro
Verkaeufer: Judith
Besucher pro Stunde: 95

3: Flammkuchenstand:
Preis pro 100g: 0.92 Euro
Verkaeufer: Kristin
Besucher pro Stunde: 57

4: Weihnachtsartikelstand:
Verkaeufer: Sandro
Besucher pro Stunde: 27

Welchen Stand moechten Sie besuchen?
4
Guten Tag!
```

```
Unsere Artikel sind:
0: Lichterkette (8.46 Euro)
1: Tasse (6.55 Euro)
2: Schlitten (2.52 Euro)
3: Windlicht (5.86 Euro)
4: Schal (0.59 Euro)
5: Handschuhe (9.92 Euro)
6: Holzpuzzle (3.36 Euro)
7: Seife (10.0 Euro)
Welchen Artikel moechten Sie kaufen?
5
Handschuhe wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
true
Unsere Artikel sind:
0: Lichterkette (8.46 Euro)
1: Tasse (6.55 Euro)
2: Schlitten (2.52 Euro)
3: Windlicht (5.86 Euro)
4: Schal (0.59 Euro)
5: ausverkauft
6: Holzpuzzle (3.36 Euro)
7: Seife (10.0 Euro)
Welchen Artikel moechten Sie kaufen?
4
Schal wird eingepackt. Viel Spass damit!
Darf es sonst noch etwas sein?
false
10.51 Euro, bitte.
Stand 4 wurde verschoben und wird jetzt von 13 Passanten 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()`.

Aufgabe 6 (Deck 6): (Codescape)

Schließen Sie das Spiel Codescape ab, indem Sie den letzten Raum auf Deck 6 auf eine der drei möglichen Arten lösen. Genießen Sie anschließend das Outro.

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.