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

Tutoraufgabe 1 (Überblickswissen):

a) Wie kann die Nutzung von Interfaces dabei helfen, die Entwicklung eines größeren Programms auf mehrere Entwickler*innen zu verteilen?

b) Welches Problem kann auftreten, wenn man zu viele `default`-Implementierungen in Interfaces nutzt?

c) Warum sind `default`-Implementierungen in Interfaces manchmal dennoch sinnvoll?

Tutoraufgabe 2 (Überschreiben, Überladen und Verdecken (Video)):

Betrachten Sie die folgenden Klassen:

Listing 1: X.java

```
1  public class X {
2      public int a = 23;
3
4      public X(int a) {                          // Signatur: X(I)
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
4      public Y(double a) {                       // Signatur: Y(D)
5          this((float) (a - 1));
6      }
7
8      public Y(float a) {                        // Signatur: Y(F)
9          super(a);
10         this.a = a;
11     }
12
13     public void f(int i, X o) { }              // Signatur: Y.f(IX)
14     public void f(int i, Y o) { }              // Signatur: Y.f(IY)
15     public void f(long lo, X o) { }            // Signatur: Y.f(LX)
16 }
```

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, wenn die `main`-Methode der Klasse `Z` ausgeführt wird. Verwenden Sie hierzu keinen Computer, sondern nur die aus der Vorlesung bekannten Angaben zum Verhalten von Java. Benutzen 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. Notieren Sie auch die von Java implizit aufgerufenen Konstruktoren. 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.

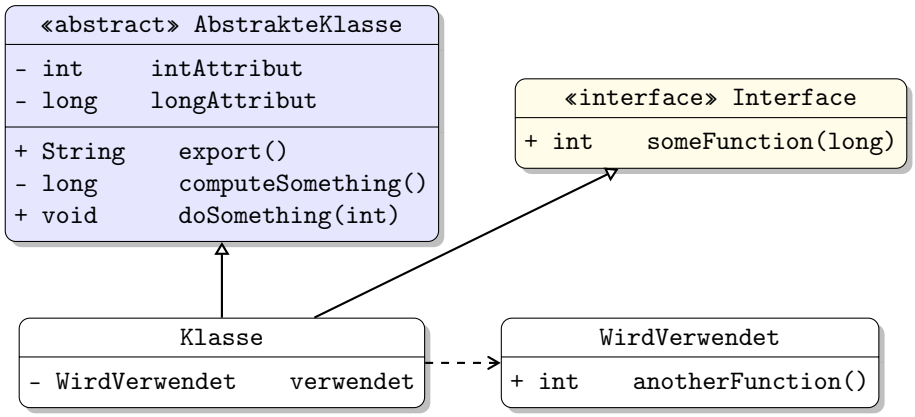
b) Geben Sie für die mit `(1)-(9)` markierten Aufrufe der Methode `f` in der Klasse `Z` jeweils an, welche Variante der Funktion von Java verwendet wird. Geben Sie hierzu die jeweilige Signatur an.

Tutoraufgabe 4 (Klassenhierarchie (Video)):

In dieser Aufgabe soll ein Weihnachtsmarkt modelliert werden.

- 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 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. 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 erhöht werden soll.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für einen Weihnachtsmarkt. Notieren Sie keine Konstruktoren, Getter oder Setter (bis auf `getPreisPro100g` und `getName`). 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 `-->` 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.

Tutoraufgabe 6 (Programmieren mit Klassenhierarchien):

In dieser Aufgabe soll es um Softwaretests gehen. Nehmen Sie folgende Situation an: bei einem Programmierwettbewerb soll ein Programm geschrieben werden, welches zwei Zahlen miteinander multipliziert. Sie haben eine Vielzahl an Einreichungen, und möchten diese automatisch auf Korrektheit testen. Um das Entwickeln einer dafür geeigneten Software und die Rahmenbedingungen des Wettbewerbs soll es in dieser Aufgabe gehen. Die Wettbewerbsbeiträge haben die Form einer Klasse, die denen von Ihnen gegebenen Einschränkungen entspricht.
Im Folgenden werden wir wieder teilweise von Abstraktion sprechen, und damit eine (evtl. abstrakte) Klasse, ein Interface oder einen Record meinen. Wählen Sie die jeweils geeignetste Variante. Ebenso sprechen wir nur von Vererbung, auch dann, wenn die Implementierung eines Interfaces möglich wäre. Machen Sie sich auch über sinnvolle Modifikatoren (wie z.B. `public`, `private`, `protected`, `final`, `sealed`, `non-sealed`, etc.) Gedanken.

a) Erstellen Sie zunächst eine Abstraktion `Identifiable`, bei der jede Klasse, die von ihr erbt, die Methode `String getName()` bereitstellt. Diese wird von den eingereichten Programmen und von den Tests genutzt werden, um einen menschlich lesbaren Namen anzugeben.

b) Jedes eingereichte Programm muss die Methode `int calculate(int x, int y)` implementieren, die das Ergebnis einer Multiplikation zweier Zahlen zurückliefern soll. Außerdem soll, wie bereits erwähnt, jedes Programm auch die Methoden von `Identifiable` bereitstellen.

Erstellen Sie eine Abstraktion `Program`, von dem jede Einreichungen erben sollte, um diese Bedingungen zu erfüllen.

c) Wir wollen verschiedene Arten von Tests auf den Programmen laufen lassen. Schreiben Sie eine Abstraktion `Test`, von der alle zukünftigen Tests erben werden. Die Abstraktion `Test` stellt die Methode `TestResult runTest(Program p)` bereit. `TestResult` ist dabei eine weitere Abstraktion, die Sie erstellen sollen. `TestResult` besitzt ein Boolesches Attribut `error` und ein Attribut `message` vom Typ `String`. Sobald ein `TestResult` erstellt worden ist, wird keine Änderung mehr an den Attributen vorgenommen, lediglich ein Zugriff auf die gespeicherten Werte ist nötig.

Weiterhin stellt `Test` die Methode von `Identifiable` bereit. Jeder `Test` hat auch ein `String`-Attribut `identifier`, das bei der Erstellung eines `Test` gesetzt werden muss und das als Standardrückgabewert von `getName()` dienen soll.

d) Wir möchten uns beim Behandeln von Tests auf zwei Arten von Tests beschränken: `PerformanceTest` und `FunctionalTest`. Andere Arten von Tests möchten wir ausschließen, modifizieren Sie `Test` entsprechend.

`PerformanceTest` ist eine konkrete Klasse, die bereits implementiert ist. In der `runTest`-Methode eines `PerformanceTest` wird lediglich die Zeit gemessen, die das übergebene Programm benötigt. Diese wird als `message`-Teil eines `TestResult`s zurückgegeben, der `error`-Wert ist bei einem `PerformanceTest` immer `false`. Die Eingabe, für die das Programm getestet wird, wird bei Erstellung des Tests festgelegt und danach nicht mehr geändert. Außerdem soll `PerformanceTest` auf eine eindeutige Weise durchgeführt werden, es soll daher keine Unterklassen geben. Es fehlen noch sinnvolle Modifikatoren, ergänzen Sie diese.

`FunctionalTest` ist lediglich als eine Überkategorie für die Tests gedacht, die die Programme auf Korrektheit überprüfen werden. Diese ist ebenfalls, bis auf Modifikatoren, bereits implementiert.

e) Nun geht es darum, konkrete Tests zu schreiben. Versuchen Sie, sinnvolle Kategorien von Eingaben zu finden, und für jede dieser Kategorien eine Klasse zu erstellen, die von `FunctionalTest` erbt. Jeder diese Testklassen soll dann in der `runTest`-Methode für mindestens eine Eingabe das Programm auswerten und das Ergebnis auf Richtigkeit überprüfen. Beispielhafte Einreichungen für den Wettbewerb finden Sie in den Klassen `MultiplyX` mit $X \in \{1, \ldots, 5\}$.

Würde es bei dem Wettbewerb um das Halbieren einer Zahl gehen, wären bspw. gerade und ungerade Zahlen mögliche Eingabekategorien und die Test-Klasse, die für gerade Zahlen zuständig ist, könnte das Programm mit der Eingabe 4 laufen lassen und überprüfen, ob 2 zurückgeliefert wird

Bei einem Fehler soll ein `TestResult` mit einem wahren `error`-Wert und einer aussagekräftigen Nachricht zurückgegeben werden, andernfalls mit einem unwahren `error`-Wert und beliebiger Nachricht.

f) Die Klasse `TestManager` ist vorgegeben, in deren `main`-Methode die `Program`-Objekte erzeugt werden und in einem Array gespeichert werden. Danach wird ein Array von Testobjekten erstellt. Momentan befinden sich in dem Array nur die Performance Tests, ergänzen Sie die von Ihnen implementierten Tests.

Schließlich müssen Sie noch die markierten Zeilen in der Methode `runTests` der Klasse `TestManager` ergänzen und können dann die Programme testen lassen. Drei der fünf Programme sind fehlerhaft, finden Ihre Tests die problematischen Programme?

g) Aus der Vorlesung kennen Sie formale Methoden, wie den Hoare-Kalkül, um ein Programm zu verifizieren. Wie unterscheidet sich eine solche Herangehensweise von Tests?