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 9.12.2019, 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 (Entwurf einer Klassenhierarchie):

In dieser Aufgabe sollen Sie einen Teil der Tierwelt modellieren.

- Ein Tier kann ein Säugetier, ein Wurm oder ein Insekt sein. Jedes Tier hat ein Alter.
- Ein spezielles Merkmal der Säugetiere ist ihr Fell. Für jedes Säugetier ist somit die Anzahl der Haare pro Quadratzentimeter Haut bekannt.
- Verschiedene Wurmarten haben im Allgemeinen wenig gemeinsam. Jeder Wurm hat jedoch eine bekannte Länge in Zentimetern.
- Alle Insekten haben einen Chitinpanzer. Bekannt ist, wie viel Druck in Pascal der Chitinpanzer eines Insektes aushalten kann.
- Menschen sind Säugetiere. Sie sind der Meinung, dass Intelligenz eines ihrer besonderen Merkmale sei. Deswegen ist der IQ jedes Menschen bekannt.
- Bandwürmer sind Würmer. Sie haben die Angewohnheit, Menschen zu befallen. Ihr vielleicht wichtigstes Merkmal ist ihr Wirt, ein Mensch, ohne den sie nicht lange überleben können.
- Bienen und Ohrwürmer sind Insekten. Das heißt insbesondere, dass Ohrwürmer keine Würmer sind.
- Bienen stechen Säugetiere, wenn sie sich bedroht fühlen.
- Ohrwürmer verfügen über Zangen, deren Größe in Millimeter in der Welt der Ohrwürmer von großer Bedeutung ist. Folglich ist die Zangengröße jedes Ohrwurms bekannt. Außerdem verwend(et)en Menschen Ohrwürmer als Medizin zur Behandlung von Erkrankungen der Ohren eines Menschen.

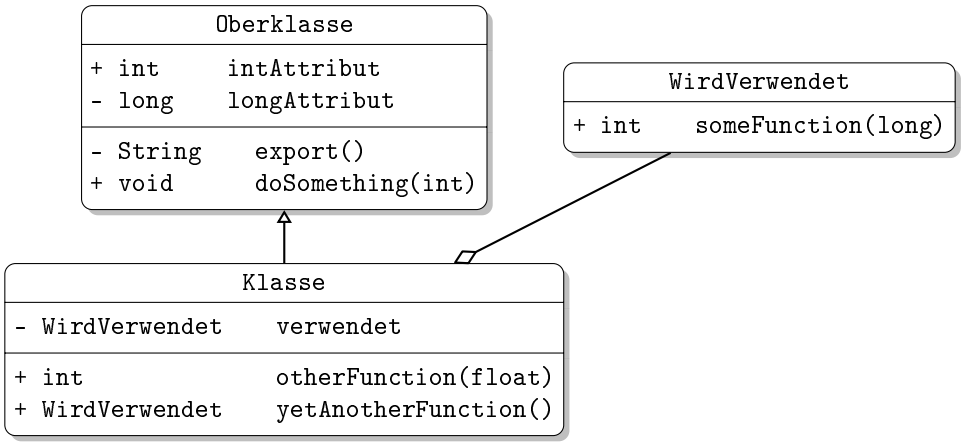

Abbildung 1: Graphische Notation zur Darstellung von Klassen.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Tieren. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden. Ergänzen Sie außerdem geeignete Methoden, um die Behandlung von Ohrenerkrankungen, den Bandwurm-Befall und den Bienenstich abzubilden.
Verwenden Sie hierbei die Notation aus Abb. 1. 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`) und $A \diamond\!\!- B$, dass A den Typ B in den Typen seiner Attribute oder in den Ein- oder Ausgabeparametern seiner Methoden verwendet. Benutzen Sie ein `-` um `private` und ein `+` um `public` abzukürzen.
Tragen Sie keine vordefinierten Klassen (String, etc.) oder Pfeile dorthin in Ihr Diagramm ein.

Aufgabe 2 (Entwurf einer Klassenhierarchie): (6 Punkte)

In dieser Aufgabe sollen Sie einen Teil der Schiffahrt modellieren.

- Ein Schiff kann ein Segelschiff oder ein Motorschiff sein. Jedes Schiff hat eine Länge und eine Breite in Metern und eine Anzahl an Besatzungsmitgliedern.
- Ein Segelschiff zeichnet sich durch die Anzahl seiner Segel aus.
- Die wichtigste Kennzahl eines Motorschiffs ist die PS-Stärke des Motors.
- Ein Kreuzfahrtschiff ist ein Motorschiff. Es hat eine Anzahl an Kabinen und kann das Schiffshorn ertönen lassen.
- Eine Yacht ist ein Segelschiff. Yachten können in Privatbesitz sein oder nicht.
- In einem aufwendigen Verfahren kann eine Yacht zu einem Kreuzfahrtschiff umgebaut werden.
- Schlepper sind Motorschiffe mit einer Anzahl von Schleppseilen. Ein Schlepper kann ein beliebiges Schiff ziehen.
- Frachter sind Motorschiffe. Sie enthalten eine Ladung, die aus einer Sammlung von Containern besteht. Außerdem sind sie durch die Größe ihres Treibstofftanks in Litern gekennzeichnet. Ein Frachter kann mit Containern be- und entladen werden, wobei beim Entladen alle Container vom Frachter entfernt werden.
- Ein Container zeichnet sich durch seine Zieladresse und das Gewicht seines Inhaltes aus. Zudem kann der Inhalt gefährlich sein oder nicht.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Schiffen. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden, falls dies sinnvoll ist. Ergänzen Sie außerdem geeignete Methoden, um das Beladen, Entladen, das Umbauen, das Ziehen und das Erklingen lassen des Horns abzubilden.
Verwenden Sie hierbei die Notation aus der entsprechenden Tutoriumsaufgabe.

Tutoraufgabe 3 (Überschreiben, Überladen und Verdecken):

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. 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. 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.

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

Betrachten Sie die folgenden Klassen:

Listing 4: A.java

```
1  public class A {
2      public final String x;
3
4      public A() {                                    // Signatur: A()
5          this("written in A()");
6      }
7
8      public A(int p1) {                              // Signatur: A(int)
9          this("written in A(int)");
10     }
11
12     public A(String x) {                            // Signatur: A(String)
13         this.x = x;
14     }
15
16     public void f(A p1) {                           // Signatur: A.f(A)
17         System.out.println("called A.f(A)");
18     }
19 }
```

Listing 5: B.java

```
1  public class B extends A {
2      public final String x;
3
4      public B() {                                    // Signatur: B()
5          this("written in B()");
6      }
7
8      public B(int p1) {                              // Signatur: B(int)
9          this("written in B(int)");
10     }
11
12     public B(A p1) {                                // Signatur: B(A)
13         this("written in B(A)");
14     }
15
16     public B(B p1) {                                // Signatur: B(B)
17         this("written in B(B)");
18     }
19
20     public B(String x) {                            // Signatur: B(String)
21         super("written in B(String)");
22         this.x = x;
23     }
24
25     public void f(A p1) {                           // Signatur: B.f(A)
26         System.out.println("called B.f(A)");
27     }
28
29     public void f(B p1) {                           // Signatur: B.f(B)
30         System.out.println("called B.f(B)");
31     }
32 }
```

Listing 6: C.java

```
1  public class C {
2      public static void main(String[] args) {
3                                                                  // a)
4          A v1 = new A(100);                                      // (1)
5          System.out.println("v1.x: " + v1.x);
6
7          A v2 = new B(100);                                      // (2)
8          System.out.println("v2.x: " + v2.x);
9          System.out.println("((B) v2).x: " + ((B) v2).x);
10
11         B v3 = new B(v2);                                       // (3)
12         System.out.println("((A) v3).x: " + ((A) v3).x);
13         System.out.println("v3.x: " + v3.x);
14
15         B v4 = new B();                                         // (4)
16         System.out.println("((A) v4).x: " + ((A) v4).x);
17         System.out.println("v4.x: " + v4.x);
18
19                                                                 // b)
20         v1.f(v1);                                               // (1)
21         v1.f(v2);                                               // (2)
22         v1.f(v3);                                               // (3)
23         v2.f(v1);                                               // (4)
24         v2.f(v2);                                               // (5)
25         v2.f(v3);                                               // (6)
26         v3.f(v1);                                               // (7)
27         v3.f(v2);                                               // (8)
28         v3.f(v3);                                               // (9)
29     }
30 }
```

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. Notieren Sie auch die von Java implizit aufgerufenen Konstruktoren. 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)`-`(9)` 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 5 (Programmieren in Klassenhierarchien):

In einer kleinen Stadt kommt es gehäuft zu Diebstählen. Sperren Sie die Diebe weg! Verwenden Sie hierbei die Hilfsklasse `Zufall` (aus dem `RWTHmoodle`-Lernraum "Programmierung (Übung - Tutorium)"), 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.

a) Schreiben Sie die Klasse `Buerger`, welche die Einwohner der Stadt modelliert. Jeder Bürger hat einen Namen (als `String`-Attribut), der an den Konstruktor übergeben wird und der bei einem Aufruf der `toString`-Methode zurückgegeben wird. Da der Name eines Bürgers nicht änderbar sein soll, definieren Sie nur eine `get`-Methode, aber keine `set`-Methode für den Namen. Weiterhin hat jeder Bürger die Methode `boolean hatDiebesgut()`, welche `false` zurückgibt. Außerdem kann jeder Bürger über die Methode `void aktion(Buerger[] buerger)` eine Aktion ausführen, welche sich ggf. auf andere Bürger in der Stadt auswirkt.

b) Ein reicher Bürger wird durch die 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. Ein reicher Bürger hat kein Diebesgut. Die Aktion (implementiert in der Methode `void aktion(Buerger[] buerger)`) 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 auf 0 initialisiert wird. Somit hat der Konstruktor nur den Parameter `name`. Die Methode `boolean hatDiebesgut()` soll zurückgeben, ob das Diebesgut größer als 0 ist. Bei einem Aufruf der Methode `void aktion(Buerger[] buerger)` hat der Dieb 5 Versuche, um Bürger zu bestehlen. In jedem Versuch soll ein Bürger aus dem Array `buerger` 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[] buerger)`, 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. Er hat kein Diebesgut. Bei einem Aufruf der Methode `void aktion(Buerger[] buerger)` sucht der Polizist bei den Bürgern im Array `buerger` 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[] buerger` besitzt. Der Konstruktor dieser Klasse bekommt das Argument `int anzahl` 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 25% Diebe, reiche Bürger, Polizisten und Gefangene gibt. Der Reichtum eines reichen Bürgers soll zufällig gewählt werden und zwischen 1 und 1000 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.

Ein Lauf des Programms könnte beispielsweise die folgende Ausgabe erzeugen:

```
Gefangener Felix aergert sich im Gefaengnis.
Gefangener Victoria aergert sich im Gefaengnis.
Dieb Toni sucht nach Diebesgut.
Dieb Toni klaut Lisa 475 Euro.
Reicher Buerger Matthias besticht einen Politiker mit 186 Euro.
Reicher Buerger Tobias besticht einen Politiker mit 194 Euro.
Dieb Toni sucht nach Diebesgut.
Dieb Toni klaut Mira 238 Euro.
Reicher Buerger Lisa besticht einen Politiker mit 234 Euro.
Dieb Toni sucht nach Diebesgut.
Dieb Toni klaut Tobias 234 Euro.
Polizist Clemens geht auf Verbrecherjagd.
Polizist Clemens entlarvt Dieb Toni.
Dieb Toni wurde eingesperrt.
Polizist Clemens geht auf Verbrecherjagd.
```

Hinweise:

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

Aufgabe 6 (Programmieren in Klassenhierarchien): (2+3.5+5+2.5+4 = 17 Punkte)

Wenn jemand einen Text schreibt, dann werden Fehler gemacht. Um eine versehentliche Änderung einfach rückgängig machen zu können, haben gängige Textverarbeitungsprogramme eine "Rückgängig" Operation (*undo*). Um diese implementieren zu können, muss das Programm jede Änderung am Textdokument sehr kontrolliert durchführen, sodass es möglich ist, die vorherige Version wiederherzustellen.

a) Erstellen Sie die Klasse `TextDocument`, welche den aktuellen Inhalt eines Textdokuments als `String`-Attribut enthält. Für ein gegebenes `TextDocument` soll der Inhalt nicht änderbar sein. Legen Sie daher nur einen Getter `getContent` an, jedoch keinen Setter. Fügen Sie außerdem einen Konstruktor hinzu, welcher den Inhalt als Parameter erhält und das Attribut entsprechend belegt. Erstellen Sie weiterhin eine Methode `undo`, welche einfach das aktuelle `TextDocument` zurückgibt.

Erstellen Sie nun die Klasse `ModifiedTextDocument`, welche `TextDocument` erweitert. Diese Klasse stellt ein Textdokument dar, zu welchem eine vorherige Version bekannt ist. Sie hält eine Referenz auf die vorherige Version in einem Attribut. Der Konstruktor von `ModifiedTextDocument` erhält sowohl den aktuellen Inhalt als `String`, als auch die vorherige Version als `TextDocument`. Die Klasse `ModifiedTextDocument` überschreibt die Methode `undo` der Klasse `TextDocument` und gibt die vorherige Version zurück.

b) Nun wollen wir zu der Klasse `TextDocument` Methoden hinzufügen, um aus einem gegebenen `TextDocument` ein neues (geändertes) `ModifiedTextDocument` zu erstellen.

Die Methode `TextDocument noop()` erstellt ein neues `ModifiedTextDocument`-Objekt, mit dem aktuellen `TextDocument`-Objekt als vorherige Version. Der neue Inhalt ist derselbe wie der aktuelle Inhalt.

Die Methode `TextDocument replaceTextSection(int beginIndex, int endIndex, String replacement)` erstellt ein neues `ModifiedTextDocument`-Objekt, mit dem aktuellen `TextDocument`-Objekt als vorherige Version. Der neue Inhalt wird aus dem aktuellen Inhalt erzeugt, indem der Textabschnitt von Position `beginIndex` bis Position `endIndex - 1` durch den Text `replacement` ersetzt wird.

Die Methode `TextDocument addTextAt(int position, String addition)` erstellt ein neues `ModifiedTextDocument`-Objekt, mit dem aktuellen `TextDocument`-Objekt als vorherige Version. Der neue Inhalt wird aus dem aktuellen Inhalt erzeugt, indem an der Position `position` der Text `addition` eingefügt wird.

Die Methode `TextDocument removeTextSection(int beginIndex, int endIndex)` erstellt ein neues `ModifiedTextDocument`-Objekt, mit dem aktuellen `TextDocument`-Objekt als vorherige Version. Der neue Inhalt wird aus dem aktuellen Inhalt erzeugt, indem der Textabschnitt von Position `beginIndex` bis Position `endIndex - 1` entfernt wird.

Hinweise:

- Nutzen Sie die Methode `String substring(int beginIndex, int endIndex)` aus der Klasse `String`, um von einem gegebenen String den Teilstring von Position `beginIndex` bis Position `endIndex - 1` zu extrahieren. Der zweite Parameter kann hierbei weggelassen werden, um den Teilstring bis zum Ende zu extrahieren. So wertet etwa `"asdf".substring(1, 3)` zu `"sd"` aus und `"asdf".substring(2)` wird zu `"df"` ausgewertet.
- Im letzten Aufgabenteil finden Sie eine Beispielausgabe für die Ausführung dieser Methoden.

c) Um Änderungen effektiv kontrollieren zu können, ist es oft sinnvoll, diese nicht direkt auszuführen, sondern die Änderung selbst als ein Objekt im Programm zu hinterlegen, um diese bei Bedarf ausführen zu können.

Legen Sie dazu die Klasse `Operation` an. Fügen Sie die Methode `TextDocument apply(TextDocument current)` hinzu, welche eine Operation auf dem übergebenen `TextDocument` ausführt und das dabei erzeugte `TextDocument` zurückgibt. In der Klasse `Operation` führt die `apply`-Methode die leere Operation (`noop`) auf dem `TextDocument current` aus.

Erstellen Sie nun vier Unterklassen von `Operation`, welche die `apply`-Methode überschreiben, sodass je eine der Methoden `undo`, `replaceTextSection`, `addTextAt` und `removeTextSection` auf dem `TextDocument current` ausgeführt wird. Einige dieser vier Methoden erwarten Parameter. Erstellen Sie für die entsprechende Unterklasse einen Konstruktor und speichern Sie diese Parameter in Attributen

der Unterklasse, sodass Sie in der `apply`-Methode die Parameterwerte aus den Attributen der Unterklasse entnehmen können. Beispielsweise soll für die Methode `addTextAt` eine Unterklasse von `Operation` namens `AddTextAtOperation` erstellt werden, deren Konstruktor die Parameter `int position` und `String addition` erhält und diese in entsprechenden Attributen der Klasse `AddTextAtOperation` zwischenspeichert. Außerdem überschreibt `AddTextAtOperation` die Methode `apply` der Klasse `Operation` und führt in dieser die Methode `addTextAt` auf dem übergebenen `TextDocument` aus. Dabei werden die in den Attributen gespeicherten Werte als Parameter der Methode `addTextAt` verwendet.

d) Da wir nun jede Operation, welche auf einem `TextDocument` ausgeführt werden kann, in einem Objekt des Typs `Operation` kapseln können, wollen wir dies nutzen, um zu jeder möglichen Operation einen Beschreibungstext zu generieren.

Ergänzen Sie die Klasse `Operation` um die Methode `String getDescription()`. Beim Aufruf soll diese Methode den Text `"does not modify the document"` zurückgeben. Überschreiben Sie diese Methode in allen vier Unterklassen und generieren Sie je eine Beschreibung, wie in der Beispielausgabe im letzten Aufgabenteil.

e) Zum Schluss wollen wir einen Beispieldurchlauf ausführen. Implementieren Sie dazu die Klasse `Launcher` mit einer `main`-Methode, welche zunächst ein Array vom Typ `Operation[]` anlegt und darin die folgenden fünf Operationen in dieser Reihenfolge ablegt:

(0) Hinzufügen des Texts `"Hello Aachen!"` an der Stelle 0

(1) Ersetzen des Textabschnitts von Zeichen 6 bis 12 durch den Text `"World"`

(2) Rückgängig machen der vorherigen Operation

(3) Ersetzen des Textabschnitts von Zeichen 0 bis 5 durch den Text `"Goodbye"`

(4) Entfernen des Textabschnitts von Zeichen 14 bis 15

Anschließend erstellt die `main`-Methode ein leeres `TextDocument` und läuft mit einer Schleife über das Array von Operationen. In jedem Schleifendurchlauf wird zunächst der Inhalt des aktuellen `TextDocument`s auf der Konsole ausgeben, dann wird die Beschreibung der aktuellen Operation ausgegeben und anschließend wird das aktuelle `TextDocument` durch das `TextDocument` ersetzt, welches von der aktuellen Operation erzeugt wird, wenn diese mit dem aktuellen `TextDocument` als Parameter ausgeführt wird. Nach Beendigung der Schleife wird noch einmal der Inhalt des aktuellen `TextDocument`s ausgegeben.

Das Ausführen der `main`-Methode sollte nun folgende Ausgabe produzieren:

```
adds the following text at position 0: Hello Aachen!
Hello Aachen!
replaces the text section from 6 to 12 by: World
Hello World!
reverts the previous operation
Hello Aachen!
replaces the text section from 0 to 5 by: Goodbye
Goodbye Aachen!
removes the text section from 14 to 15
Goodbye Aachen
```

Hinweise:

- Nutzen Sie die Methode `System.out.println(String output)` um einen Text auf der Konsole auszugeben.
- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung.

Aufgabe 7 (Deck 6): (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 Montag, den 9.12.2019, um 12:00 Uhr abschicken.