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

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.

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

Betrachten Sie die folgenden Klassen:

Listing 1: 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 2: 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 3: 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.

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.