

Allgemeine Hinweise:

- Die **Deadline** zur Abgabe der Hausaufgaben ist am **Donnerstag, den 04.12.2025, um 14 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine*n Abgabepartner*in zu finden. Es darf **nur ein*e** Abgabepartner*in die Abgabe hochladen. Diese*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als **.zip**-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass uns Ihre Hiwis mit Feedback versehen können. Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von **.java**-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen dürfen nicht mit abgegeben werden. Stellen Sie sicher, dass Ihr Programm von **javac** in der **Version 25** akzeptiert wird. Generell sollten alle Programme für alle Eingaben terminieren, solange in der Spezifikation (bzw. der Aufgabenstellung) nicht explizit etwas anderes verlangt wird!
- Einige Hausaufgaben müssen im Spiel **Codescape** gelöst werden. Klicken Sie dazu im Lernraum rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Übungsaufgabe 1 (Überblickswissen):

- a) Viele Vorteile der Vererbung zwischen zwei Klassen A und B1 lassen sich auch erzielen, ohne dass diese Klassen tatsächlich voneinander erben. So ist es beispielsweise möglich, die Klasse B1 in eine Klasse B2 zu überführen, welche sich genau wie B1 verhält, ohne jedoch von A zu erben.

Wir passen die Klasse B1 an, indem wir die **extends**-Klausel weglassen und stattdessen ein neues Attribut **a** vom Typ **A** in die Klasse B2 einfügen und diesem Attribut ein neues Objekt der Klasse A zuweisen. Wenn in B1 Attribute oder Methoden der Oberklasse A verwendet werden, müssen diese sich in B2 nun auf das Attribut **a** beziehen.

Wir haben also die *Vererbungsbeziehung* zwischen A und B1 durch eine *Nutzungsbeziehung* zwischen A und B2 ersetzt. Was sind die Vor- und Nachteile dieser beiden Varianten?

```

class A {
    int i;
    void m() {}
}

class B1 extends A {
    void foo() {
        m();
        i += 1;
    }
}

class B2 {
    A a = new A();
    void foo() {
        a.m();
        a.i += 1;
    }
}
    
```

b) Betrachten Sie den folgenden Java Code:

```

public class A {
    public final int n = 1;
    public int m(A p) {
        return 3;
    }
}

public class B extends A {
    public final int n = 2;
    public int m(A p) {
        return 4;
    }
    public int m(B p) {
        return 5;
    }
}

public class Launcher {
    public static void main() {
        B b = new B();
        A ab = new B();
        IO.println(ab.n);
        IO.println(ab.m(b));
        IO.println(b.m(ab));
        IO.println(b.m(b));
    }
}

```

Es werden die Zahlen 1, 4, 4 und 5 ausgegeben.

- (i) Welche Rolle spielen folgende Typen bei der Ausgabe der ersten Zahl?
 - (1) Der deklarierte Typ der Variablen `ab` (`A`).
 - (2) Der Laufzeittyp des in `ab` gespeicherten Objekts (`B`).
- (ii) Welche Rolle spielen folgende Typen bei der Ausgabe der zweiten, dritten und vierten Zahl?
 - (1) Der deklarierte Typ der Variablen vor dem Punkt (`A`, `B` und `B`).
 - (2) Der Laufzeittyp des Objekts, welches in der Variablen vor dem Punkt (`B`, `B` und `B`) gespeichert ist.
 - (3) Der deklarierte Typ des Parameters (`B`, `A` und `B`).
 - (4) Der Laufzeittyp des Objekts, welches als Parameter übergeben wird (`B`, `B` und `B`).

c) Betrachten Sie den folgenden Java Code:

```

class A {}
class X extends A {}
class Y extends A {}
class Z extends A {}

public void match(A a) {
    switch(a) {
        case X x -> IO.println("Laufzeittyp ist X");
        case Y y -> IO.println("Laufzeittyp ist Y");
        case Z z -> IO.println("Laufzeittyp ist Z");
        default -> IO.println("Laufzeittyp ist weder X, noch Y, noch Z");
    }
}

```

Im `switch` Statement ist ein `default` Fall vorhanden. Fehlt dieser, so erzeugt Java einen Compilerfehler, da möglicherweise die Variable `a` ein Objekt mit einem Laufzeittyp enthält, welcher weder `X`, noch `Y`, noch `Z` ist.

Falls in der Praxis jedoch klar ist, dass dieser `default` Fall nie ausgeführt werden kann, so wäre es wünschenswert ihn direkt wegzulassen. So wird die Lesbarkeit des Codes verbessert.

Auf welche Art und Weise muss obiger Code ergänzt werden, sodass der `default` Fall weggelassen werden kann?

d) Zusätzlich zu den Klassen aus dem vorherigen Aufgabenteil ist nun noch der folgende Record gegeben:

```
public record KlassenPaar(A x, A y) {}
```

Geben Sie ein Beispiel für eine `switch`-Anweisung mit Record Patterns, welche unbenannte Patterns und unbenannte Pattern Variablen benutzt.

Übungsaufgabe 2 (Entwurf einer Klassenhierarchie):

In dieser Aufgabe sollen Sie einen Teil der Möbelwelt modellieren.

- Ein Möbelstück hat ein Gewicht und ein bestimmtes Material. In dieser Aufgabe betrachten wir Sitzmöbel, Tische und Behältnismöbel.
- Ein Tisch ist ein Möbelstück mit einer Anzahl von Tischbeinen und einer bestimmten Länge und einer bestimmten Breite.
- Ein Esstisch ist ein Tisch, welcher einer gewissen Anzahl von Personen Platz bietet. Er kann aber auch um eine bestimmte Länge und eine bestimmte Breite verlängert werden.
- Ein Sitzmöbelstück hat eine gewisse Anzahl von Sitzplätzen.
- Ein Stuhl ist ein Sitzmöbelstück. Stühle können an einen Tisch gestellt werden. Dies funktioniert nicht immer.
- Ein Schaukelstuhl ist ein besonderer Stuhl. Er kann schaukeln.
- Ein Behältnismöbelstück hat ein Volumen und kann gefüllt werden.
- Ein Schrank ist ein Behältnismöbelstück mit einer bestimmten Anzahl von Fächern. Einen Schrank kann man öffnen.
- Kleiderschränke sind Schränke, die über einen Spiegel verfügen können und aufgeräumt werden können.
- Ein Bücherregal ist ein Behältnismöbelstück, welches natürlich Bücher enthält. Ein Buch kann entnommen werden, wenn es im Regal steht.
- Ein Buch hat einen Titel und eine Anzahl von Seiten. Bücher können gelesen werden.

Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für den oben dargestellten Sachverhalt. Notieren Sie keine Konstruktoren. Um Schreibarbeit zu sparen, brauchen Sie keine Selektoren anzugeben. Sie müssen nicht markieren, ob Klassen **sealed** oder ob Attribute **final** sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen zusammengefasst werden, falls dies sinnvoll ist. Ergänzen Sie außerdem geeignete Methoden, um das Verlängern eines Esstisches, das Stellen eines Stuhls an einen Tisch, das Schaukeln eines Schaukelstuhls, das Füllen eines Behältnismöbelstücks, das Öffnen eines Schranks, das Aufräumen eines Kleiderschranks, das Entnehmen eines Buchs aus einem Regal sowie das Lesen eines Buchs zu modellieren.

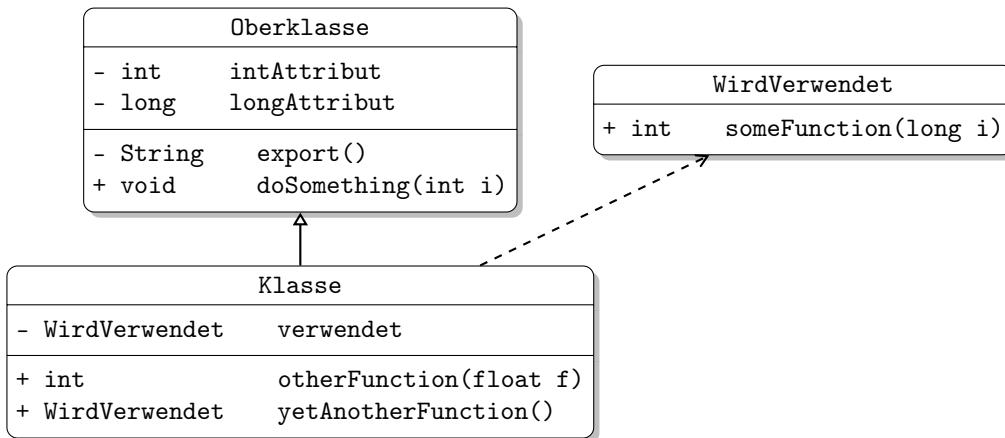


Abbildung 1: Graphische Notation zur Darstellung von Klassen.

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 $B \rightarrow A$, dass A die Oberklasse von B ist (also `class B extends A`) und $A \rightarrow B$, 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.

Hausaufgabe 3 (Entwurf einer Klassenhierarchie):**(11 Punkte)**

In dieser Aufgabe sollen Sie einen Teil der Schifffahrt modellieren.

- Die wichtigste Eigenschaft eines Hafens ist sein Name.
- 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 Schiff hat seinen Liegeplatz in einem Hafen.
- Ein Segelschiff zeichnet sich durch die Anzahl seiner Segel aus.
- Die wichtigste Kennzahl eines Motorschiffs ist die PS-Stärke des Motors. Der Motor kann außerdem ein Dieselmotor oder ein Elektromotor sein.
- 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.
- 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 seinen Eigentümer, seine Zieladresse und das Gewicht seines Inhalts aus. Zudem kann der Inhalt gefährlich sein oder nicht.
- In einem aufwendigen Verfahren kann ein Frachter zu einem Kreuzfahrtschiff umgebaut werden.

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. Sie müssen nicht markieren, ob Klassen **sealed** oder ob Attribute **final** sein sollen. 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. Tragen Sie keine vordefinierten Klassen (**String**, etc.) oder Pfeile dorthin in Ihr Diagramm ein.

Verwenden Sie hierbei die Notation aus der Übungsaufgabe 2.

Übungsaufgabe 4 (Überschreiben, Überladen und Verdecken):

Betrachten Sie die folgenden Klassen:

```

public class X {
    public int a = 23;
    public X(int a) { // Signatur : X(I)
        super();
        this.a = a;
    }
    public X(float x) { // Signatur : X(F)
        this((int) (x + 1));
    }
    public void f(int i, X o) { } // Signatur : X.f(IX)
    public void f(long lo, Y o) { } // Signatur : X.f(LY)
    public void f(long lo, X o) { } // Signatur : X.f(LX)
}

public class Y extends X {
    public float a = 42;
    public Y(double a) { // Signatur : Y(D)
        this((float) (a - 1));
    }
    public Y(float a) { // Signatur : Y(F)
        super(a);
        this.a = a;
    }
    public void f(int i, X o) { } // Signatur : Y.f(IX)
    public void f(int i, Y o) { } // Signatur : Y.f(IY)
    public void f(long lo, X o) { } // Signatur : Y.f(LX)
}

public class Z {
    public static void main() {
        X xx1 = new X(42); // (1)
        IO.println("X. a:" + xx1.a);
        X xx2 = new X(22.99f); // (2)
        IO.println("X. a:" + xx2.a);
        X xy = new Y(7.5); // (3)
        IO.println("X. a:" + ((X) xy).a);
        IO.println("Y. a:w" + ((Y) xy).a);
        Y yy = new Y(7); // (4)
        IO.println("X. a:" + ((X) yy).a);
        IO.println("Y. a:" + ((Y) yy).a);

        int i = 1;
        long lo = 2;
        xx1.f(i, xy); // (1)
        xx1.f(lo, xx1); // (2)
        xx1.f(lo, yy); // (3)
        yy.f(i, yy); // (4)
        yy.f(i, xy); // (5)
        yy.f(lo, yy); // (6)
        xy.f(i, xx1); // (7)
        xy.f(lo, yy); // (8)
    }
}

```

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 Konstruktoraufufe 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)-(8) 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.

Hausaufgabe 5 (Überschreiben, Überladen und Verdecken): (4 + 6 = 10 Punkte)

Betrachten Sie die folgenden Klassen:

```

public class A {
    public static int x = 0;
    public int y = 7;

    public A () {                                     // Signatur: A()
        this(x);
        x++;
    }

    public A (int x) {                             // Signatur: A(I)
        x = x + 2;
        y = y - x;
    }

    public A (double x) {                         // Signatur: A(D)
        y += x;
    }

    public void f(int i, A o) { }                  // Signatur: A.f(IA)
    public void f(Long lo, A o) { }                // Signatur: A.f(LA)
    public void f(double d, A o) { }                // Signatur: A.f(DA)
}

public class B extends A {
    public float x = 1.5f;
    public int y = 1;

    public B () {                                 // Signatur: B()
        x++;
    }

    public B (float x) {                         // Signatur: B(F)
        super(x);
        super.y++;
    }

    public void f(int i, B o) { }                // Signatur: B.f(IB)
    public void f(int i, A o) { }                // Signatur: B.f(IA)
    public void f(long lo, A o) { }              // Signatur: B.f(LA)
}

public class C {
    public static void main() {
        // a)
        A a1 = new A(5);                         // (1)
        IO.println (A.x);
        IO.println (a1.y);
        A a2 = new A();                          // (2)
        IO.println (A.x);
        IO.println (a2.y);
        B b = new B();                           // (3)
        IO.println (A.x);
        IO.println (b.x);
        IO.println (((A) b).y);
        IO.println (b.y);
        A ab = new B(3);                        // (4)
        IO.println (ab.y);
        IO.println (((B) ab).y);
    }
}

```

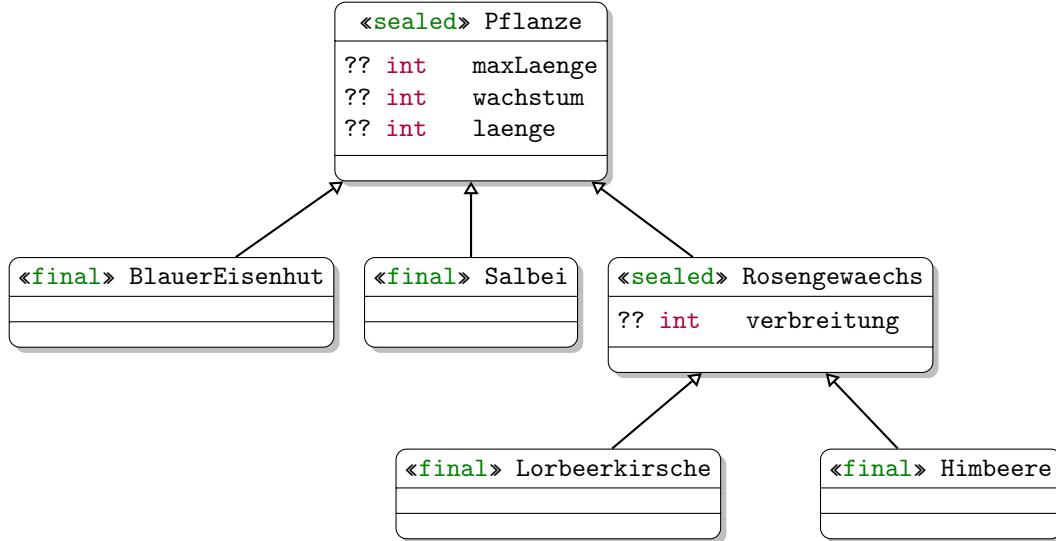
```
// b)
int i = 1;
long l = 2;
double d = 3.0;
a1.f(i, a1);           // (1)
a1.f(l, ab);          // (2)
b.f(i, (B) ab);       // (3)
b.f(d, ab);           // (4)
ab.f(i, a1);          // (5)
ab.f(i, b);           // (6)
}
}
```

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.

- 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.
- Geben Sie für die mit (1)-(6) 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.

Übungsaufgabe 6 (Programmieren in Klassenhierarchien):

In dieser Aufgabe modellieren wir 4 verschiedene Pflanzenarten: LorbeerKirsche, Himbeere, Salbei und BlauerEisenhut. Hierfür nutzen wir folgende Klassenhierarchie:



- Implementieren Sie die Klassen **Pflanze**, **Rosengewaechs**, **LorbeerKirsche**, **Himbeere**, **Salbei** und **BlauerEisenhut** anhand dem oben stehenden Klassendiagramm. Nutzen Sie hierbei die Schlüsselwörter **sealed** und **final**, damit es keine weiteren Klassen geben kann, die von der Klasse **Pflanze** (oder den anderen) erben. Jede Pflanze hat eine maximale Länge (**maxLaenge**), eine aktuelle Länge (**laenge**) und eine Wachstumsrate (**wachstum**), die alle als **int** gespeichert werden. Ein **Rosengewaechs** hat zusätzlich noch eine bestimmte Verbreitungsrate (**verbreitung**), die auch als **int** gespeichert wird. Nutzen Sie passenden Zugriffsmodifikatoren für die Attribute, so dass auf diese innerhalb der Hierarchie zugegriffen werden kann. Erstellen Sie außerdem entsprechende **get**-Methoden für jedes Attribut.
- Implementieren Sie für die Klassen **Pflanze** und **Rosengewaechs** jeweils einen Konstruktor, der einen Parameter für jedes Attribut dieser Klasse bekommt und diesen dann entsprechend setzt. Hierbei soll darauf geachtet werden, dass keine **Pflanze** mit einer größeren Länge als ihrer maximalen Länge erstellt werden soll. Falls also der Parameter für die momentane Länge größer ist als der für die maximale Länge, dann soll die momentane Länge stattdessen auf 0 gesetzt werden. Für die restlichen Klassen soll jeweils ein Konstruktor erstellt werden, der den Konstruktor der Oberklasse mit den folgenden entsprechenden Attributen aufruft:
 - **Salbei** hat die maximale Länge 6 und die Wachstumsrate 1.
 - Ein **BlauerEisenhut** hat keine maximale Länge, daher soll diese auf den größtmöglichen **int**-Wert gesetzt werden, und die Wachstumsrate 1.
 - Eine **Himbeere** hat die maximale Länge 10, die Wachstumsrate 1 und die Verbreitungsrate 2.
 - Eine **LorbeerKirsche** hat die maximale Länge 20, die Wachstumsrate 2 und die Verbreitungsrate 3.
 Außerdem startet jede dieser vier Pflanzen mit der aktuellen Länge 1.
- Jede Pflanze hat eine Methode **waessern()**, die die Länge der Pflanze vergrößert und keinen Rückgabewert hat. Generell soll beim Ausführen zu der Länge die Wachstumsrate addiert werden, allerdings nur, solange das Ergebnis nicht über die maximale Länge hinaus wächst. In diesem Fall wächst die Pflanze nur bis zu der maximalen Länge heran. Zusätzlich wird bei Rosengewächsen noch die Verbreitungsrate mit der Wachstumsrate multipliziert bevor wir diese zur momentanen Länge addieren. Implementieren Sie eine Methode **waessern()** passend in der Hierarchie.
- Jede Pflanze hat eine Methode **schnieden(int x)**, allerdings gibt es für jede Pflanzenart leichte Unterschiede. Implementieren Sie eine Methode **schnieden(int x)** in der Klasse **Pflanze**, die Sie in den

anderen Klassen passend überschreiben. Implementieren Sie die Methode mit den folgenden Spezifikationen:

- Für Objekte der Klassen **Pflanze**, **Salbei** und **Himbeere** wird die Länge der Pflanze um den Parameter **x** verringert.
- Bei einer **LorbeerKirsche** ist das Schneiden leider sehr anstrengend, weswegen wir hier die Länge nur um die Hälfte von **x** (abgerundet) verringern können.
- Für Objekte der Klassen **Rosengewaechs** und **BlauerEisenhut** wird die Länge immer auf 1 heruntergeschnitten, egal welchen Wert der Parameter **x** enthält.

Beachten Sie, dass eine Pflanze keine negative Länge haben kann. Falls also bei einer **Pflanze** mit der **Laenge** 3 die Methode **schneiden** mit dem Parameter 5 aufgerufen wird, dann soll danach die **Pflanze** eine **laenge** von 0 haben. Beachten Sie außerdem, dass ein **Rosengewaechs** beim Schneiden keine Länge dazugewinnen kann, falls es vor dem Aufruf der Methode eine Länge von 0 besitzt.

Hinweise:

- Bei einem negativen Eingabeparameter **x** darf sich Ihre Methode beliebig verhalten.

e) Erstellen Sie ein Record **Pflanzenpaar**, welches einfach ein Paar von 2 Pflanzen darstellt.

f) Wir wollen in dieser Teilaufgabe dem Kleingärtner Otto bei der Auswahl seiner Pflanzen helfen. Jeden Tag erhält dieser zwei verschiedene Pflanzen und muss dann entscheiden, welche Pflanze er davon einzupflanzen soll. Danach muss die einzupflanzende Pflanze je nach Art unterschiedlich behandelt werden. Implementieren Sie die Methode **Pflanze auswahl(Pflanzenpaar pair)** in der Klasse **Feld**. Nutzen Sie zur Auswahl der Pflanze eine **switch**-Anweisung mit Record Patterns, wie Sie es in der Vorlesung gelernt haben (d.h. eine Anweisung der Form **switch(pair) { ... }**). In dieser muss mindestens eine unbenannte Pattern Variable oder ein unbenannter Pattern vorkommen. Es gelten folgende Bestimmungen bei der Auswahl der Pflanze:

- **Rosengewaechse** werden immer vor anderen Pflanzen präferiert. Diese werden dann solange bewässert, bis sie zur maximal erreichbaren Länge heran gewachsen sind. Falls Otto eine **Himbeere** ausgewählt hat, will er sie nach dem Bewässern direkt kürzer schneiden und ruft die Methode **schneiden** mit dem Parameter 1 auf.
- **BlauerEisenhut** soll vor **Salbei** präferiert werden, falls die Länge vom **Salbei** größer oder gleich 5 ist.
- Falls sich Otto nicht entscheiden kann, dann wählt er immer die erste Pflanze des Paars aus.

Zum Schluss gibt die Methode **Pflanze auswahl(Pflanzenpaar pair)** die ausgewählte Pflanze zurück. Falls das übergebene **Pflanzenpaar** oder die Pflanzen innerhalb des **Pflanzenpaars null** sind, darf sich die Methode beliebig verhalten.

Hinweise:

- Sie benötigen mindestens Java 23 um eine **switch**-Anweisung mit Record Patterns und unbenannte Pattern Variablen zu implementieren!
- Sie finden in der Klasse **Feld** auch eine **main**-Methode. Diese können Sie zum Testen Ihrer Implementation nutzen. Wenn alles richtig implementiert ist, sollte es die folgenden 5 Pflanzen ausgeben in genau dieser Reihenfolge:
 - Auswahl:Himbeere@..., Laenge:9
 - Auswahl:LorbeerKirsche@..., Laenge:20
 - Auswahl:LorbeerKirsche@..., Laenge:20
 - Auswahl:BlauerEisenhut@..., Laenge:1
 - Auswahl:Salbei@..., Laenge:1

Hausaufgabe 7 (Programmieren in Klassenhierarchien): (3 + 7 + 10 + 2.5 + 2 + 4.5 = 29 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 jeweils 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 Beispieldruck im letzten Aufgabenteil.
- e) Ergänzen Sie die Klassen um die Modifikatoren `sealed`, `non-sealed` und `final`, sodass zur Compilezeit bereits feststeht, dass es neben den oben genannten Operationen keine weiteren geben kann und auch keine weiteren Unterklassen der oben genannten Operationen existieren. Außerdem soll es neben dem `ModifiedTextDocument` keine weitere Unterklasse von der `TextDocument`-Klasse geben, aber Unterklassen von der `ModifiedTextDocument`-Klasse selbst wollen wir nicht ausschließen.
- f) 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 `TextDocuments` auf der Konsole ausgegeben, 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 `TextDocuments` 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 `IO.println(String output)` um einen Text auf der Konsole auszugeben.
- Berücksichtigen Sie in der gesamten Aufgabe die Prinzipien der Datenkapselung.

Hausaufgabe 8 (Deck 7):**(Codescape)**

Schließen Sie das Spiel Codescape ab, indem Sie die letzten Missionen von Deck 7 lösen. Genießen Sie anschließend das Outro. Dieses Deck enthält keine für die Zulassung relevanten Missionen.

Hinweise:

- Es gibt drei 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.