

JAVA 101




Inhaltsverzeichnis:

1. Version – erste Klassen und Methoden erstellen
2. Version – Erweiterung der bisherigen Klassen und Verknüpfung untereinander
3. Version – Vererbung
4. Version – abstrakte Klassen und Methoden
5. Version – Scanner, Zufallszahlen & instanceof Operator
6. Version – Einlesen von Textdateien & Wrapper Classes

Musterlösung (keine Garantie):

<https://github.com/mittey68/prog-tutorium>

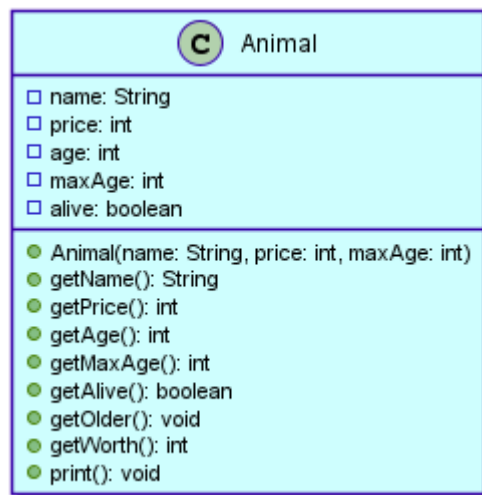
Hinweise & Tipps:

- Erstelle jede Übung als eigenes Package. So behält jede Version ihre Lauffähigkeit, trotz Änderungen in der neuen Version
- UML-Legende:
 -  = private
 -  = public
 -  = protected
- ArrayList, Color, Scanner, etc müssen bei Verwendung in die jeweiligen Klassen importiert werden. Dazu bietet Eclipse in der entsprechenden Fehlermeldung einen QuickFix an.

Übungen

1. Version – erste Klassen und Methoden erstellen
 - a. Erstelle die Klasse „Animal“ wie abgebildet
 - i. Der Konstruktor soll alle(!) Attribute initialisieren (Hinweis: Es dürfen keine toten Tiere erstellt werden und Tiere sind beim Erstellen immer 0 Jahre alt)
 - ii. getName(), getPrice(), getAge(), getMaxAge() & getAlive() sind Get-Methoden für die Attribute
 - iii. getOlder() soll das Alter des Tieres um 1 erhöhen, außer das Maximalalter ist erreicht. Dann soll alive auf „falsch“ gesetzt werden.
 - iv. getWorth() soll der Wert des Tiers berechnen. Dabei sind Tiere gleich oder jünger 5 Jahre die Hälfte ihres Preises wert. Tiere die älter als 5 Jahre sind haben den gleichen Wert wie ihr Preis.
 - v. Print() soll eine Konsolenausgabe wie unten aufgeführt ergeben.
 - b. Erstelle die ausführbare Klasse Main
 - i. Erstelle 3 Tiere
 - ii. Gib die Tiere auf der Konsole aus
 - iii. Lass die Tiere altern
 - iv. Gib die Tiere erneut aus

c. UML Diagramm



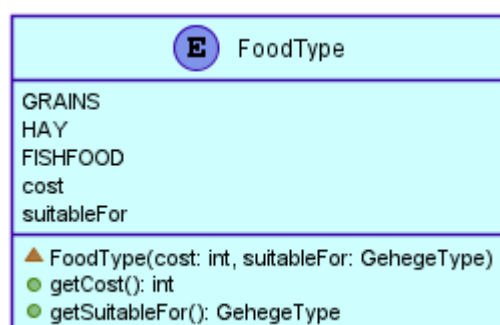
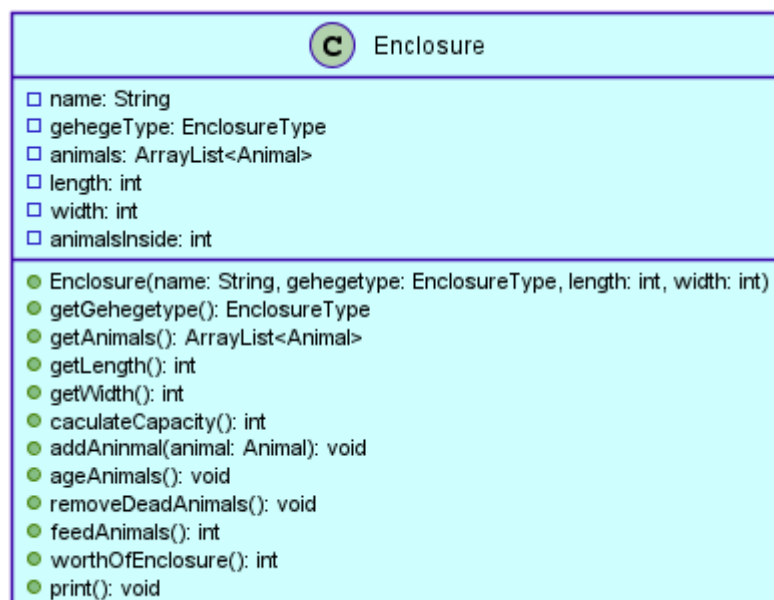
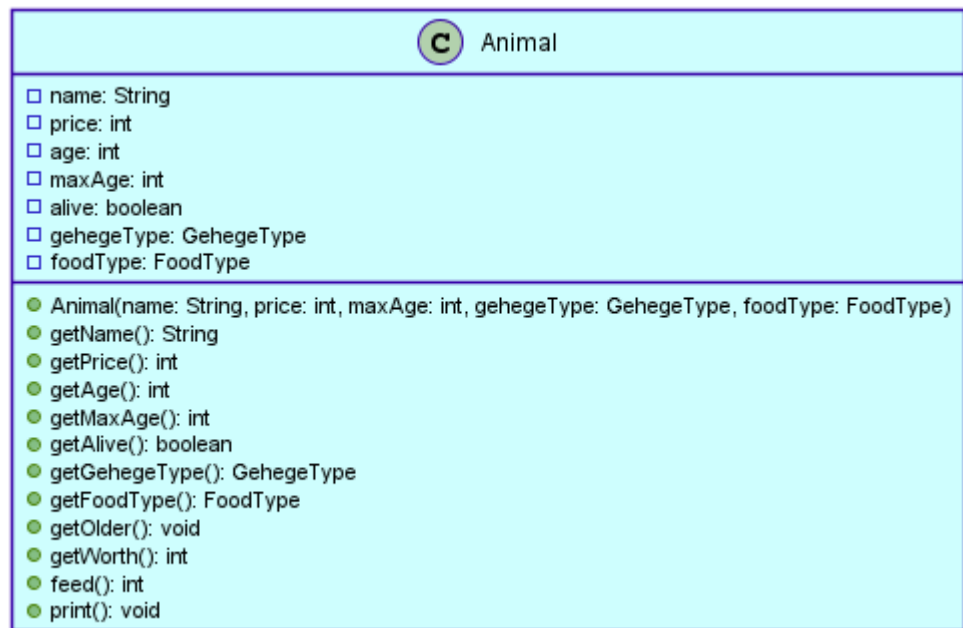
d. Konsolenausgabe

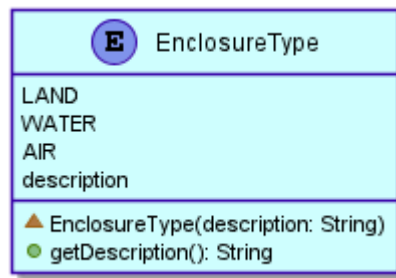
```
Tier:
- Name: Pferd
- Preis: 1000
- Aktueller Wert: 500
- Alter: 0
- Geschätzte Lebenserwartung: 10
- Am Leben?: true
Tier:
- Name: Vogel
- Preis: 60
- Aktueller Wert: 30
- Alter: 0
- Geschätzte Lebenserwartung: 7
- Am Leben?: true
Tier:
- Name: Fisch
- Preis: 30
- Aktueller Wert: 15
- Alter: 0
- Geschätzte Lebenserwartung: 5
- Am Leben?: true
Tier:
- Name: Pferd
- Preis: 1000
- Aktueller Wert: 1000
- Alter: 6
- Geschätzte Lebenserwartung: 10
- Am Leben?: true
Tier:
- Name: Vogel
- Preis: 60
- Aktueller Wert: 30
- Alter: 1
- Geschätzte Lebenserwartung: 7
- Am Leben?: true
Tier:
- Name: Fisch
- Preis: 30
- Aktueller Wert: 15
- Alter: 5
- Geschätzte Lebenserwartung: 5
- Am Leben?: false
```

2. Version – Erweiterung der bisherigen Klassen und Verknüpfung untereinander

- a. Verändere die Klasse Animal wie angegeben
 - i. Füge die beiden neuen Attribute ein und initialisiere sie (Hinweis: Beachte die Enumeration Hilfsklassen unten, oder schreibe sie anhand von den UML Diagrammen selbst)
 - ii. Erstelle die neuen Getter
 - iii. Die Methode feed() soll die Kosten des jeweiligen FoodType aus dem Attribut foodType ausgeben
 - iv. Verändere die print() Methode entsprechend der Konsolenausgaben
- b. Erstelle die Klasse Enclosure wie angegeben
 - i. Der Konstruktor soll alle Attribute initialisieren (Hinweis: animalsInside sagt aus, wie viele Tiere aktuell im Gehege sind und ist beim Erstellen daher 0)

- ii. `getEnclosureType()`, `getAnimals()`, `getLength()`, `getWidth()` sind die entsprechenden Getter für die Attribute
 - iii. Die Methode `calculateCapacity()` gibt die Maximalkapazität des Geheges wieder. Diese berechnet sich durch $(\text{länge} * \text{breite}) / 1000$.
 - iv. `addAnimals()` soll das übergebene Animal in das Gehege hinzufügen, vorausgesetzt die Maximalkapazität ist nicht erreicht & das Gehege ist für das entsprechende Tier geeignet (Beachte `gehegeType`). Die Fehlermeldungen können aus der Konsolenausgabe herausgelesen werden.
 - v. `ageAnimals()` soll alle Tiere im Gehege altern lassen & die unten angegebene Hilfsmethode `removeDeadAnimals()` aufrufen.
 - vi. `feedAnimals()` soll die Gesamtkosten für das Futter von allen Tieren in einem Gehege zurückgeben
 - vii. `worthOfEnclosure()` soll den Gesamtwert aller Tiere eines Geheges ausgeben.
 - viii. Die `print()` Methode muss entsprechend der Konsolenausgaben angepasst werden. (Hinweis: Hier soll die `print()` Methode von `Animal` aufgerufen werden)
- c. Verändere die Main Klasse
- i. Erstelle 3 Land-, 3 Luft- und 3 Wassertiere
 - ii. Erstelle jeweils ein Land-, Luft- und Wassergehege
 - iii. Erstelle ein zu kleines Gehege mit der Länge 1 und Breite 1
 - iv. Teste die Fehlermeldungen, wenn ein falsches Gehege beim Hinzufügen verwendet wird und wenn die Maximalkapazität des Geheges nicht ausreichend ist.
 - v. Füge die Tiere ihrem jeweils richtigen Gehege hinzu
 - vi. Gib die Gehege auf der Konsole aus
 - vii. Lasse die Tiere im Gehege altern
 - viii. Gib die Tiere auf der Konsole aus & teste, ob die Alterung funktioniert.
- d. UML Diagramme





e. Konsolenausgaben

```

Dieses Gehege ist zu klein. Die Maximalkapazität von 0 ist erreicht.
Dieses Tier passt nicht in ein Luftgehege. Es benötigt ein Landgehege
Gehegenname: Pferdekoppel
Gehegetyp: LAND
Gehegekapazität: 60
Tiere im Gehege: 3
Tier:
- Name: Pferd 1
- Tierart: LAND
- Preis: 1000
- Aktueller Wert: 500
- Alter: 0
- Geschätzte Lebenserwartung: 10
- Am Leben?: true
Tier:
- Name: Pferd 2
- Tierart: LAND
- Preis: 4600
- Aktueller Wert: 2300
- Alter: 0
- Geschätzte Lebenserwartung: 10
- Am Leben?: true
Tier:
- Name: Pferd 3
- Tierart: LAND
- Preis: 300
- Aktueller Wert: 150
- Alter: 0
- Geschätzte Lebenserwartung: 10
- Am Leben?: true
Gehegenname: Vogelvoliere
Gehegetyp: AIR
Gehegekapazität: 30
Tiere im Gehege: 3
Tier:
- Name: Vogel 1
- Tierart: AIR
...
  
```

f. Hilfsklassen und -methoden

i. Enum EnclosureType

```

package zoo.v2;

public enum EnclosureType {
    /*
     * Aufzählungskonstanten
     */
    LAND("Landgehege"), WATER("Wassergehege"), AIR("Luftgehege");

    private String description; // Beschreibung des Gehegetyps

    // Konstruktor
    EnclosureType(String description) {
        this.description = description;
    }

    // Getter für Description
    public String getDescription() {
        return description;
    }
}

```

ii. Enum FoodType

```

package zoo.v2;

public enum FoodType {
    /*
     * Aufzählungskonstanten
     */
    GRAINS(5, GehegeType.AIR), HAY(3, GehegeType.LAND), FISHFOOD(10, GehegeType.WATER);

    private int cost; // Kosten des Tierfutter
    private GehegeType suitableFor; // GehegeTyp für das Futter

    // Konstruktor
    FoodType(int cost, GehegeType suitableFor) {
        this.cost = cost;
        this.suitableFor = suitableFor;
    }

    // Getter für Kosten
    public int getCost() {
        return cost;
    }

    // Getter für Gehegetyp
    public GehegeType getSuitableFor() {
        return suitableFor;
    }
}

```

iii. Hilfsmethode removeDeadAnimals()

```

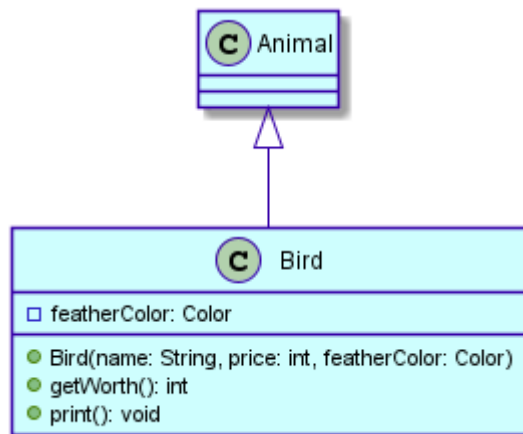
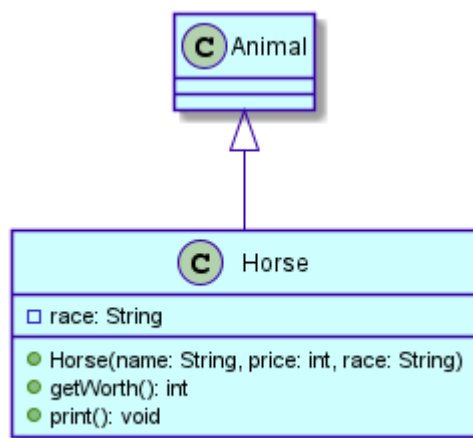
public void removeDeadAnimals() {
    for (int i = 0; i < this.animals.size(); i++) {
        Animal animal = this.animals.get(i);
        if (animal.getAlive() == false) {
            this.animals.remove(i);
            this.animalsInside--;
        }
    }
}

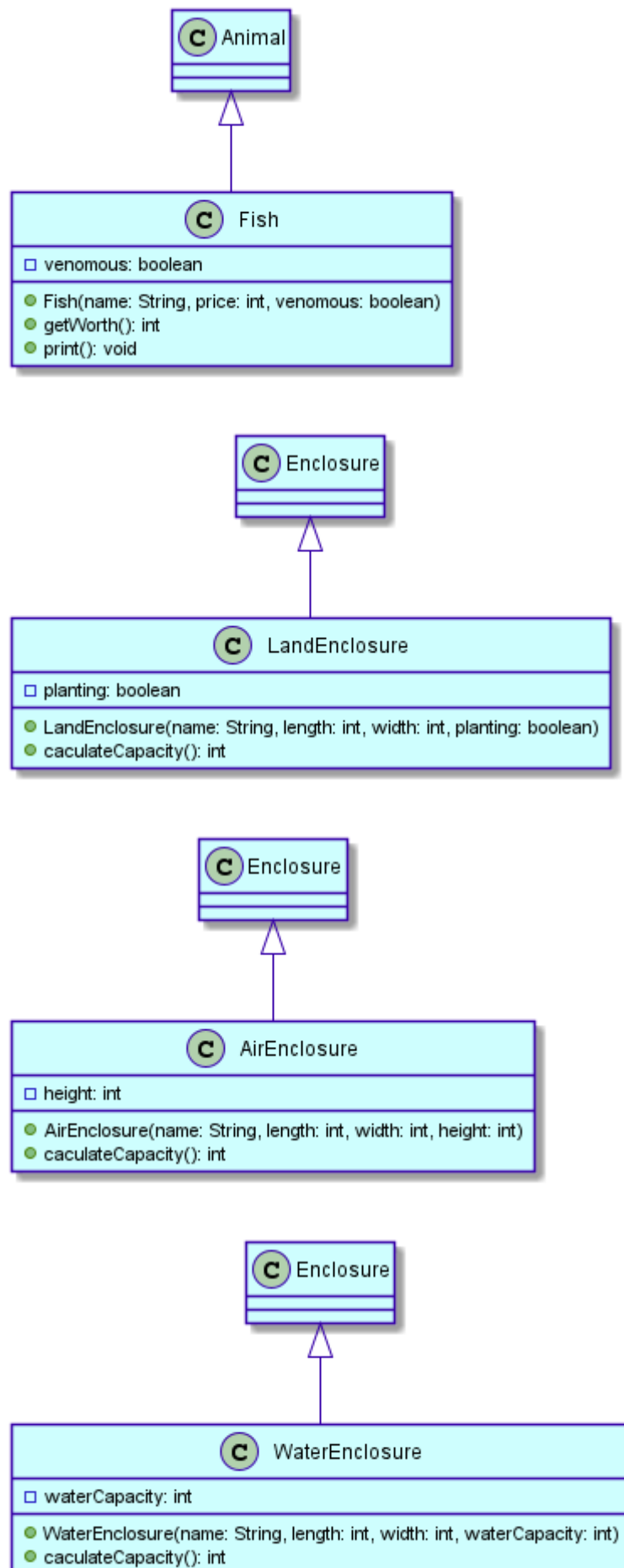
```

3. Version – Vererbung

- a. Erstelle die 3 Unterklassen der Klasse „Animal“ „Bird“, „Horse“ und Fish
 - i. Die Unterklassen sollen die Methoden print() und getWorth() der Oberklasse überschreiben
 - ii. Horse:
 1. Pferde haben das zusätzliche Attribut „race“ in welchem die Rasse des Tieres angegeben wird
 2. Der Wert eines Pferdes berechnet sich aus seinem Alter:
 - a. Pferde die 5 oder jünger sind, sind die Hälfte ihres Kaufpreises wert
 - b. Pferde die älter als 5 aber jünger als 11 sind, sind das Doppelte ihres Preises wert
 - c. Pferde die 11 oder älter sind haben den Wert ihres Preises
 3. Die Methode print() soll zusätzlich die Rasse des Pferds ausgeben
 - iii. Bird
 1. Vögel haben das zusätzliche Attribut „feathercolor“, welches Objekte der Javaklasse Color aufnehmen kann.
 2. Der Wert des Vogels wird anhand seiner Federfarbe berechnet
 - a. Gelbes Gefieder verzehnfacht den Wert gegenüber dem Kaufpreis
 - b. Grünes Gefieder verdoppelt den Wert gegenüber dem Kaufpreis
 - c. Andere Gefiederfarben haben keine Auswirkungen auf den Wert und haben somit den Kaufpreis als Wert
 3. Die Methode print() soll zusätzlich die Gefiederfarbe ausgeben
 - iv. Fish
 1. Fische haben das zusätzliche Attribut „venomous“, welches true/false aufnehmen kann.
 2. Der Wert des Fisches berechnet sich aus seiner Giftigkeit
 - a. Giftige Fische haben den dreifachen Wert ihres Preises
 - b. Ungiftige Fische haben den Wert ihres Preises
 3. Die Methode print() soll zusätzlich die Giftigkeit ausgeben
- b. Erstelle die Unterklassen der Klasse Enclosure „LandEnclosure“, „AirEnclosure“ und „WaterEnclosure“
 - i. Die Unterklassen sollen die Methode caculateCapacity() der Oberklasse überschreiben
 - ii. Landgehege
 1. Landgehege haben das zusätzliche Attribut „planting“ welches angibt, ob Bepflanzung im Gehege vorhanden ist
 2. Die Kapazität berechnet sich aus $(\text{Länge} \cdot \text{Breite} / 1.000)$ und halbiert sich, wenn Bepflanzung vorhanden ist.

- iii. Luftgehege
 - 1. Luftgehege haben als zusätzliches Attribut die Höhe.
 - 2. Die Kapazität berechnet sich aus $(\text{Länge} * \text{Breite} * \text{Höhe} / 10.000)$
- iv. Wassergehege
 - 1. Wassergehege haben als zusätzliches Attribut die Wasserkapazität
 - 2. Die Kapazität berechnet sich aus $(\text{Länge} * \text{Breite} * \text{Wasserkapazität} / 100.000)$
- c. Verändere die Main Klasse entsprechend
 - i. Erstelle statt der 9 Animals 3 Horse-, 3 Bird- und 3 Fish-Objekte
 - ii. Erstelle statt der 3 Gehege jeweils ein Land-, Wasser- und Luftgehege
- d. UML Diagramme





e. Konsolenausgabe

```
Dieses Gehege ist zu klein. Die Maximalkapazität von 0 ist erreicht.  
Dieses Tier passt nicht in ein Luftgehege. Es benötigt ein Landgehege  
Gehegenname: Pferdekoppel  
Gehegetyp: LAND  
Gehegekapazität: 30  
Tiere im Gehege: 3  
Pferd:  
- Name: Gisela  
- Rasse: Schimmel  
- Tierart: LAND  
- Preis: 1000  
- Aktueller Wert: 500  
- Alter: 0  
- Geschätzte Lebenserwartung: 10  
- Am Leben?: true  
Pferd:  
- Name: Peter  
- Rasse: Schimmel  
- Tierart: LAND  
- Preis: 4600  
- Aktueller Wert: 2300  
- Alter: 0  
- Geschätzte Lebenserwartung: 10  
- Am Leben?: true  
Pferd:  
- Name: Horst  
- Rasse: Schimmel  
- Tierart: LAND  
- Preis: 300  
- Aktueller Wert: 150  
- Alter: 0  
- Geschätzte Lebenserwartung: 10  
- Am Leben?: true  
Gehegenname: Vogelvoliere  
Gehegetyp: AIR  
Gehegekapazität: 300  
Tiere im Gehege: 3
```

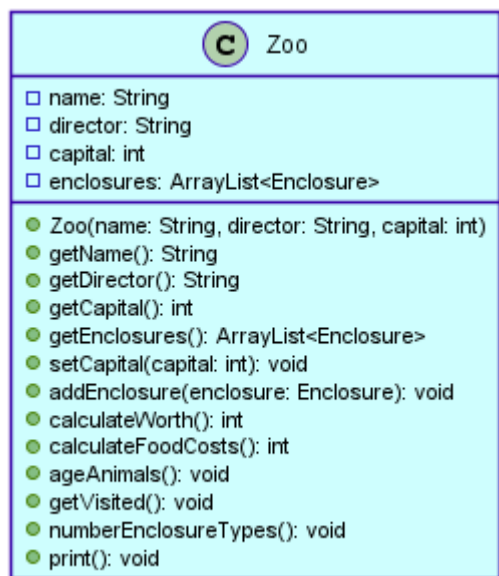
4. Version – abstrakte Klassen und Methoden

- a. Verändere die Klasse Animal
 - i. Animal soll nicht mehr instanziiert werden können
 - ii. Die Methoden print() und getWorth() sollen erzwungenermaßen von den Unterklassen implementiert werden müssen
- b. Verändere die Klasse Enclosure
 - i. Gehege soll nicht mehr instanziiert werden können
 - ii. Die Methode getCapacity() sollen erzwungenermaßen von den Unterklassen implementiert werden müssen

5. Version – Scanner, Zufallszahlen & instanceof Operator

- a. Erstelle die Klasse Zoo
 - i. Der Konstruktor soll alle Attribute initialisieren
 - ii. getName(), getDirector(), getCapital(), getEnclosures() sind die Getter der entsprechenden Attribute
 - iii. setCapital() ist der Setter des entsprechenden Attributs
 - iv. addEnclosure() soll dem Zoo ein Gehege hinzufügen

- v. `calculateWorth()` soll den Wert aller Tiere des Zoos zurückgeben
 - vi. `calculateFoodCosts()` soll die Futterkosten aller Tiere zurückgeben
 - vii. `AgeAnimals()` soll alle Tiere des Zoos altern lassen
 - viii. `getVisited()` soll den Wert aller Tiere auf das Kapital des Zoos aufaddieren, die Futterkosten vom Kapital abziehen und 1000 als Fixkosten vom Kapital abziehen. Des Weiteren sollen in dieser Methode die Tiere einmal gealtert werden.
 - ix. `numberEnclosureTypes()` soll als Hilfsmethode der `print()` Methode per Konsolenausgabe die Anzahl der jeweiligen Gehegetypen ausgeben, d.h. wie viele Land-, Luft- und Wassergehege im Zoo existieren.
 - x. `print()` soll die Konsolenausgabe wie unten aufgeführt erzeugen und dabei die Hilfsmethode `numberEnclosureTypes()` und die `print()` Methode der Klasse `Enclosure` verwenden
- b. Verändere die Main Klasse
- i. Ließ den Zoonamen per Scanner ein
 - ii. Ließ den Namen des Zoodirektors per Scanner ein
 - iii. Erstelle für das Kapital eine Zufallszahl zwischen 2000 – 10000
 - iv. Erstelle den Zoo mit diesen Angaben
 - v. Füge dem Zoo die Gehege inklusive der Tiere hinzu
 - vi. Lasse den Zoo über die Methode `getVisited()` mehrmals Besucher empfangen und prüfe auch die Alterung der Tiere und die Veränderung des Kapitals
- c. UML Diagramme



- d. Konsolenausgabe

```

Gibt bitte den Namen deines Zoos ein: Alex Zoo
Gibt deinen Namen ein: Alex
-----
Name des Zoos: Alex Zoo
Zoodirektor: Alex
Aktuelles Kapital: 8948
Anzahl unserer Gehege:
0 Landgehege
0 Luftgehege
0 Wassergehege
Unsere Tiere:
-----
Dieses Gehege ist zu klein. Die Maximalkapazität von 0 ist erreicht.
Dieses Tier passt nicht in ein Luftgehege. Es benötigt ein Landgehege
-----
Name des Zoos: Alex Zoo
Zoodirektor: Alex
Aktuelles Kapital: 8948
Anzahl unserer Gehege:
1 Landgehege
1 Luftgehege
1 Wassergehege
Unsere Tiere:
Gehegenname: Pferdekoppel
Gehegetyp: LAND
Gehegekapazität: 30
Tiere im Gehege: 3
Pferd:
- Name: Gisela
- Rasse: Schimmel
- Tierart: LAND

```

6. Version – Einlesen von Textdateien & Wrapper Classes

- a. Erstelle eine Textdatei wie unten angegeben oder lade sie aus dem Repository herunter
- b. Verändere die Main Klasse
 - i. Ergänze „throws FileNotFoundException“ im Header deiner Main-Methode
 - ii. Ließ die erstellte Datei zeilenweise ein
(Hinweis: Wenn die Textdatei im Projektverzeichnis abgelegt wird, kann die relative Pfadangabe „<Name der Textdatei>.txt“ verwendet werden)
 - iii. Jede Zeile der Datei gibt die Daten für ein Objekt der Klasse Horse an
 - iv. Erstelle die Pferde mit den eingelesenen Daten aus der txt-Datei & füge sie einem bereits erstellten Gehege hinzu
- c. Textdatei

```
Herbert;7800;Araber  
Arthus;25000;Holstein  
Lennox;40000;Haflinger
```