HTWK Leipzig, FIMN
Dipl.-Math. Dörte König

Anwendungsorientierte Programmierung II
Hinweise zur Übung7

1. Strukturierung der Klassen

Basisklasse Zootier

Oberklasse, enthält Eigenschaften und Fähigkeiten der allgemeinen Typgruppe
Eigenschaften und Fähigkeiten werden erst in den abgeleiteten Unterklassen spezifiziert
Basisklasse ZooTier muss deshalb nicht instanziierbar sein → abstrakte Klasse

```
public abstract class Zootier {}
```

Erste Ableitung: Klassen der Tierfamilien

Klassen für die Tierfamilien `Canoidea`, `Feloidea`
abstrakte Klassen, da von ihnen keine Instanzen gebildet werden müssen

```
public abstract class Canoidea extends Zootier{}
```

Zweite Ableitung: Klassen für die Tiere

aus den Tierfamilien-Klasse sind die zugehörigen Tier-Klassen abzuleiten
keine abstrakte Klassen, da sie instanziierbar sein müssen

```
public class Bernhardiner extends Canoidea{}
```

2. Interface HaustierIF

In Java ist Mehrfachvererbung nicht möglich, d.h. eine Klasse kann nur zu genau einer Oberklasse gehören, um deren Eigenschaften zu erben.
Allerdings kann eine Klasse in Java mehrere Interfaces implementieren.

Interface

Ein Interfaces stellt sicher, dass diejenigen Klassen die das Interface implementieren, bestimmte Fähigkeiten oder Eigenschaften zwingend enthalten.
Es darf normalerweise nur Konstanten und abstrakte Methoden enthalten.

Abstrakte Methoden

sind Methoden, für die im Interface (oder einer Klasse) nur die Signatur angegeben wird, die aber dort nicht implementiert sind. Um die Funktionen nutzen zu können, müssen die abstrakten Methoden in der implementierenden Klasse ausprogrammiert werden.

Interface `HaustierIF`

enthält sämtliche Methoden, die alle Haustiere besitzen sollen (siehe Klassendiagramm)

```
public interface HaustierIF
{
	String getName();
	...
}
```

Alle Methoden eines Interfaces sind automatisch `public`
→ Modifier muss nicht zwingend angegeben werden.

3. **Erweiterung der Haustierklassen**
   Attribute `name` und `laut` ergänzen
   Interface `HaustierIF` implementieren
   fehlenden Methoden aus dem Interface über die Quickfixes hinzufügen und ausprogrammieren

```
public class Beagle extends Canoidea implements HaustierIF
{
	...

	@Override
	public String getName()
	{
		return name;
	}
	...
}
```

→ siehe Klassendiagramm

Klasse Main:

4. **Listen**
   Speicherung der erzeugten Tiere in Listen

```
List katzen=new ArrayList();
```

   typisierte Listen stellen sicher, dass in der Liste nur Objekte des angegebenen Typs abgelegt werden können

```
List<Feloidea>katzen=new ArrayList<Feloidea>();
List<Feloidea>katzen=new ArrayList<>();              //ab Java7
```

   Hinzufügen neuer Objekte

```
katzen.add(new Amurtiger());
```

   Ausgabe
   in einem String mit Hilfe der `toString()`-Methode

```
System.out.println(katzen.toString());
```

   als einzelne Objekte über eine `for-each`-Schleife erfolgen

```
for(Feloidea k : katzen)
{
	System.out.println(k);
}
```

5. **Streichelzoo**
   Liste `streichelzoo`

```
List<HaustierIF>streichelzoo=new ArrayList<HaustierIF>();
```

   mit den Haustieren aus den Tierfamilien-Listen füllen
   jedes Haustier erhält dabei einen Namen

mit dem **`instanceof`-Operator** kann getestet werden, ob ein Tier ein Haustier ist (d.h. das `HaustierIF` implementiert):

```
for(Feloidea k: katzen)
{
	if(k instanceof HaustierIF)
	{ ... }
}
```

Optionale Aufgabe

1. **Klasse Gehege**

   Generics verwenden

   ```
   public class Gehege <T extends Zootier>{}      //T ist Typparameter fuer
                                                  //irgendeine Subklasse von Zootier
   ```

   Attribut:

   ```
   private ArrayList<T> wildtiere;
   ```

   Methoden:

   selbst überlegen

2. **Erweiterung der Main-Klasse**

   Anlegen der Wildtier-Gehege durch Konkretisierung des Typparameters T

   ```
   Gehege<Amurtiger>taiga=new Gehege<Amurtiger>();
   Gehege<Erdmaennchen>savanne=new Gehege<Erdmaennchen>();
   ...
   ```

   Hinzufügen der Wildtiere zum Gehege

   ```
   for(Feloidea k: katzen)
   {
   	String art= k.getClass().getSimpleName();
   	switch(art)
   	{
   		case "Amurtiger": taiga.addWildtier((Amurtiger) k);
   		break;
   		...
   	}
   }
   ```