HTWK Leipzig, FIMN
Dipl.-Math. Dörte König

Anwendungsorientierte Programmierung II
Hinweise zur Übung 6

Aufgabe 1

A. Klasse Angler – Kassenstand

Attribut Kassenstand

`Kassenstand` ist instanzunabhängiges Attribut, auf das alle Angler zugreifen können (müssen)

Deklaration:
```
protected static double kassenstand = 0;
```

Statische Methode `getKassenstand()`

Deklaration:
```
public static double getKassenstand(){}
```

Modifier `protected`

Elemente mit dem Modifier `protected` sind

- sichtbar in der eigenen und allen abgeleiteten Klassen sowie für alle nicht verwandten Klassen innerhalb desselben Paketes
- nicht sichtbar für Klassen, die in anderen Paketen definiert wurden

B. Klasse Main – Anglerverein

Anglerverein = Feld mit Instanzen der Klasse Angler

Variante 1: direkte Initialisierung

```
Angler[] angler={   new Angler("Hein","Jensen",12.45),
                    new Angler("Fiete","Holthusen",10.00),
                    //...
                };
```

Variante 2: zufällige Werte aus vorgegebenen Arrays

```
String[] vnamen={"Hein","Fiete",...};
String[] nnamen={"Hansen","Petersen",...};
double[] beitraege={10.4,15.7,...};

Angler[] pj = new Angler[n];

pj[i]= new Angler(vnamen[z.nextInt(vnamen.length)],
                  nnamen[z.nextInt(nnamen.length)],
                  beitraege[z.nextInt(beitraege.length)]);
```

Aufgabe 2

A. Vererbung

Allgemeine Regeln:

Eine Klasse B (Unterklasse) kann von einer Klasse A (Oberklasse) abgeleitet werden.
Dabei gilt:

B **erbt** alle Eigenschaften und Methoden von A
B kann **zusätzliche Eigenschaften und Methoden** deklarieren
B kann Methoden von A überschreiben (**Polymorphie**),
d.h. die Methode hat den gleichen Namen und die gleiche Signatur wie die Oberklasse, aber eine andere Funktionalität

Vererbung in Java:

Vererbung wird bei der Klassendeklaration über das **Schlüsselwort `extends`** realisiert:

```
public class KlasseB extends KlasseA {}
```

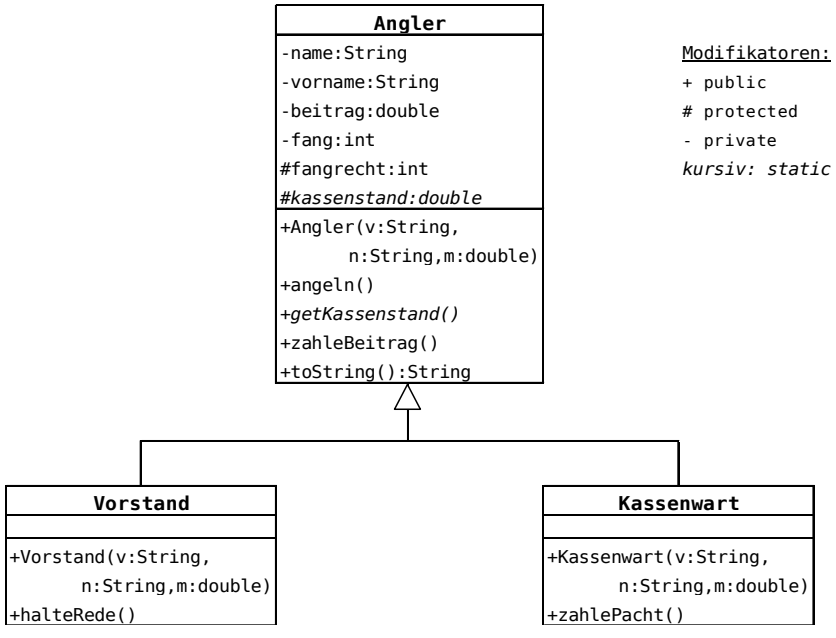
Mehrfachvererbung ist (im Gegensatz zu C++) **nicht erlaubt**, d.h. eine Unterklasse kann nur von einer Oberklasse abgeleitet werden.

Methoden von Unterklassen können auf `private` deklarierte Elemente der Oberklasse nicht direkt zugreifen. Möchte man den Zugriff ermöglichen, so muss der **Sichtbarkeitsmodifikator `protected`** verwendet werden.

Abgeleitete Klassen sollten einen **eigenen Konstruktor** haben. Dieser kann den Konstruktor seiner Oberklasse über `super()` aufrufen und muss damit nur die zusätzlich in der abgeleiteten Klasse deklarierten Attribute explizit initialisieren:

```
public KlasseB(String v, int n, double d)
{
	super(v,n);
	eigenesAttribut1 = d;
	eigenesAttribut2 = 4;
}
```

B. Klassendiagramm



C. Entwurfsmuster Singleton

Entwurfsmuster (oder Design Pattern) sind wiederverwendbare Muster, die bestimmte häufig wiederkehrende Entwurfsprobleme abdecken. Das Entwurfsmuster Singleton gewährleistet, dass von einer Klasse nur eine Instanz erzeugt werden kann.

Im Anglerverein darf es nur einen Vorstand und einen Kassenwart geben
→ Anwendung des Entwurfsmusters Singleton bei der Erzeugung der Instanzen

Beispiel :

```
public class Vorstand extends Angler
{
	//Variable für (einzige) Instanz
	private static Vorstand instance;

	//Konstruktor als private!
	//nur über die getInstance()-Methode kann
	//genau eine Instanz erzeugt werden
	private Vorstand(String v, String n, double m)
	{
		super(v, n, m);
		fangrecht=4;
	}

	//(einzige)Instanz erzeugen
	public static Vorstand getInstance (String v, String n, double m)
	{
		if (instance == null)
			instance =   new Vorstand(v,n,m);
	return instance;
	}

	//...weitere Methoden
}
```

Eine Instanz der Klasse Vorstand wird dann nicht mehr über den direkten Konstruktoraufruf erzeugt, sondern über den Aufruf der statischen Methode `getInstance()`:

```
pj[0] = Vorstand.getInstance( ... );
```

Programm beenden:

See.java:

```
public static void dispose()
{
    window.dispose();
}
```

Main.java

```
See.dispose();
```