Polymorphismus, Typen

Polymorphismus

- ist die „dritte Säule der OOP“ (Kapslung, Vererbung, Polymorphismus)
- griechisch für „Vielgestaltigkeit“

Objekte abgeleiteter Klasse können immer als Objekte der Mutterklassen betrachtet werden

```
Lebewesen menschAlsLebewesen = new Mensch();
```

Variablentyp vs. Laufzeittyp

```
Mensch mensch = new Mensch()
Variablentyp variablenbezeichner = new Laufzeittyp()
```

- Der Variablentyp bestimmt den Typ einer Variablen und dadurch deren möglichen Inhalt
- Der Laufzeittyp bezeichnet den Typ eines Objekts und kann über die Methode `.GetType()`erfragt werden

Typtest durch GetType und typeof-Operator

- durch „.GetType()“ wird der Laufzeittyp eines Objektes ermittelt
- der typeof Operator ruft die System.Type-Instanz für einen Typ ab

```
Lebewesen menschAlsLebewesen = new Mensch();

if (menschAlsLebewesen.GetType() == typeof(Lebewesen))
//false

if (menschAlsLebewesen.GetType() == typeof(Mensch))
//true
```

Typtest durch is-Operator

- der is-Operator prüft ob der Laufzeittyp mit einem angegebenen Typ kompatibel ist
- gibt „true“ zurück, wenn das Objekt auch als der erfragte Typ betrachtet werden kann / wenn eine Ableitung existiert

```
Lebewesen lebewesen = new Lebewesen();
Mensch mensch = new Mensch();

if (lebewesen is Lebewesen) //true
if (lebewesen is Mensch) //false
if (mensch is Lebewesen) //true
if (mensch is Mensch) //true
if (mensch is object) //true
```

virtuelle Member

- es wird immer auf den Member der jeweiligen Instanz zugegriffen
- die Instanz kann auch in die Basisklasse gecastet werden, der Aufruf geschieht dennoch auf die eigentliche Instanz
- in Basisklasse casten ergibt Sinn, wenn mehrere Objekte unterschiedlicher abgeleiteter Klassen verarbeiten werden

```
Lebewesen lebewesen = new Lebewesen();
lebewesen.WasBinIch();
//"Ich bin ein Lebewesen"

Mensch mensch = new Mensch();
mensch.WasBinIch();
//"Ich bin ein Lebewesen auf zwei Beinen und kann sprechen"

Lebewesen menschAlsLebewesen = (Lebewesen)mensch;
menschAlsLebewesen.WasBinIch();
//"Ich bin ein Lebewesen auf zwei Beinen und kann sprechen"
```

Ausblenden von Basisklassenmembern

- Member aus der Basisklasse können auch ausgeblendet werden
- das Schlüsselwort „new“ vor den Rückgabewert setzen und den gleichen Membernamen verwenden
- wenn eine Instanz in die Basisklasse gecastet wird, kann diese nun nicht mehr auf das neue Member zugreifen

```
public class Lebewesen
{
    public virtual string WasBinIch()
    {
        return "Ich bin ein Lebewesen";
    }
}

public class Mensch : Lebewesen
{
    public new string WasBinIch()
    {
        return "Ich bin ein Mensch";

    }
}
Mensch mensch = new Mensch();
Lebewesen menschAlsLebewesen = (Lebewesen)mensch;

mensch.WasBinIch(); //"Ich bin ein Mensch"
menschAlsLebewesen.WasBinIch(); //"Ich bin ein Lebewesen"
```

abstrakte Member und Klassen

- sind nur Signaturen und geben den Aufbau vor, können aber nicht selber initialisiert werden
- müssen von abgeleiteten Klassen überschrieben werden
- wenn ein Member abstrakt sein soll, muss die Klasse als abstrakt markiert werden

```
public abstract class Fahrzeug
{
    public abstract bool IstFahrbereit();
}

public class Fahrrad : Fahrzeug
{
    public int Luft { get; set; }

    public override bool IstFahrbereit()
    {
        return (Luft > 0);
    }
}
```

abstrakte Member und Klassen

- können nicht „sealed“ verwenden, da sie sich gegenseitig ausschließen
- eine abgeleitete Klasse muss alle abstrakten Member implementieren
- abstrakte Methoden haben keinen Körper sondern werden nur deklariert