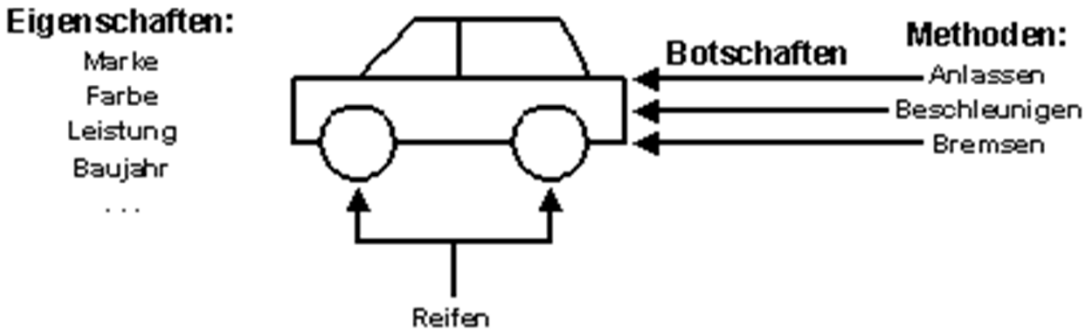
Objektorientierung



Objektorientierte Programmierung

- Wir Menschen sind gewohnt, in Objekten zu denken. In der Schule lernen wir bereits, ähnliche Formen wie Dreiecke, Kreise oder Vierecke zu jeweils einer Gruppe zusammenzufassen. Früh können wir erkennen, dass Fahrrad, Motorrad, Auto, Bus oder Lastwagen eine Gemeinsamkeit haben: Sie sind alle Fortbewegungsmittel.

Objektorientierte Programmierung

- Im Alltag ordnen Sie Birnen, Äpfel, Orangen etc. ganz selbstverständlich der Gruppe Obst zu und Salat und Kohl sind für Sie Gemüse.
- Beim Programmieren eines Onlineshops arbeiten Sie ebenso mit Objekten. Beispielsweise mit Artikeln, Kunden und Farben.
- Unser Leben, die Wirklichkeit, ist voller Objekte: Fahrzeuge, Obst, Gemüse, Artikel, Kunden und viele mehr.

Objektorientierte Programmierung

- Die Programmierer haben erkannt, dass sie Probleme dann wirklichkeitsnah lösen können, wenn sie für die objektorientierten Aufgabenstellungen auch objektorientierte Lösungen finden.
- Bei der objektorientierten Programmierung können Sie reale Objekte direkt im Programm abbilden. Dabei ist jedes programmierte Objekt durch sein Verhalten, seine Identität und seinen Zustand identifizierbar - und dem wirklichen Objekt im realen Leben sehr ähnlich.
- Sie können durch die Objektorientierung einmal programmierte Objekte in verschiedenen Anwendungen wieder verwenden.

Warum Objektorientierung?

- Die Idee hinter der Objektorientierung ist es menschliche Organisationsmethoden in der realen Welt in Objekten nachzubilden.

Realität vs. Programmierung

- Fenster aufmachen
- Physik: Kraft ausüben
- prinzipiell auch Unvorhergesehenes möglich, wie „Fenster einschlagen“

vs.

- Fenster.aufmachen ()
- Modell: „Eigenverantwortung des Objekts“
- prinzipiell nie Unvorhergesehenes möglich, weil das Objekt vorher Reaktion auf Befehl trainiert/programmiert bekommen muss

Tabu-Karten

- In welche Kategorien lassen sich die Eigenschaften der beiden Objekte einteilen?

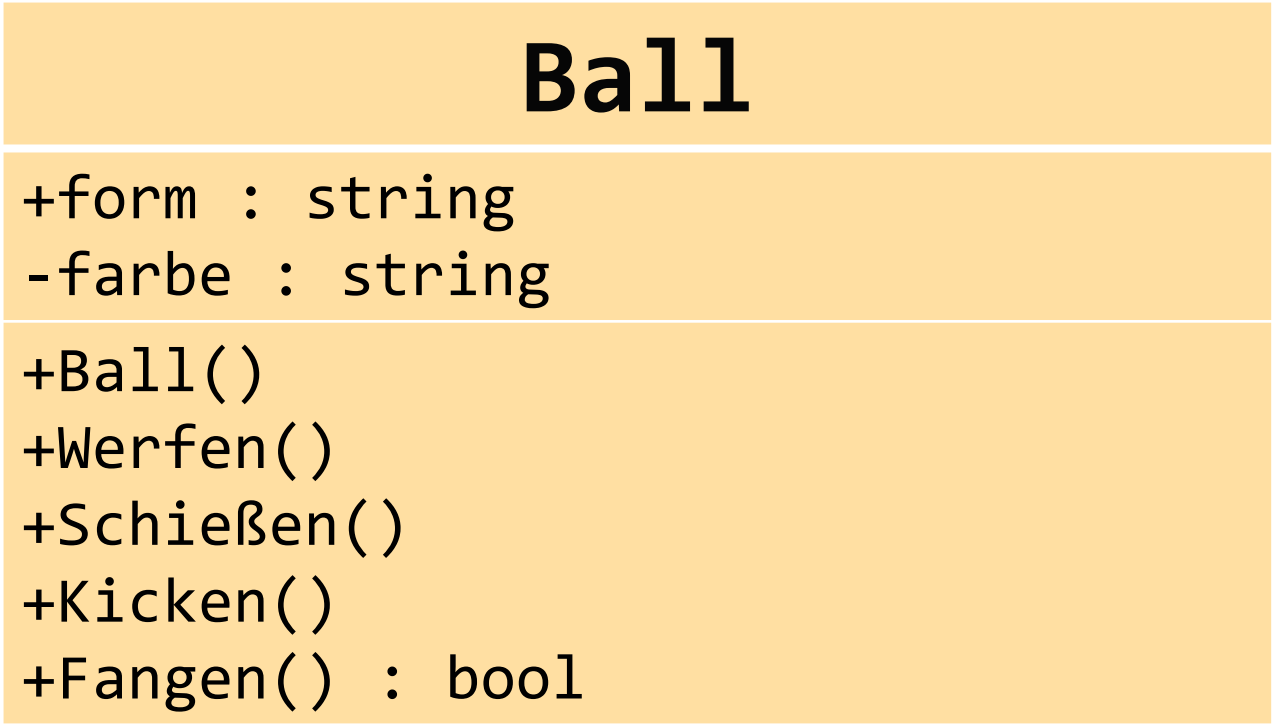
Ball	Stift
rund	lang
kugelförmig	dünn
werfen	bunt
schießen	malen
kicken	zeichnen
fangen	schreiben

Tabu-Karten

- Attribute und Methoden

Ball	Stift
rund	lang
kugelförmig	dünn
werfen	bunt
schießen	malen
kicken	zeichnen
fangen	schreiben

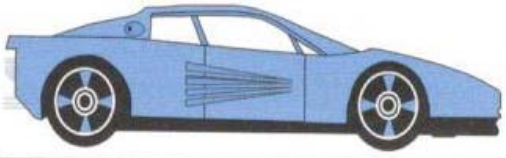
Klasse Ball in UML-Darstellung



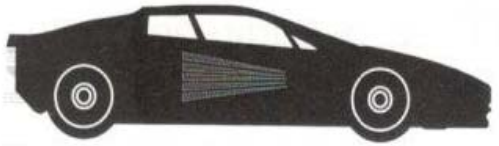
Klassen, Objekte und Instanzen

- **Klassen** kann man grob als “Bauplan” ansehen, mit dem Bauplan selbst kann man nichts machen, aber man kann den Bauplan nutzen, um etwas herzustellen, z. B. ein Objekt.
- **Objekte** sind praktisch Gegenstände die mittels eines Bauplanes (Klasse) gebaut werden können.
- Objekte und Instanzen können für den hausgebrauch als das gleiche angesehen werden, denn: **Ein Objekt ist die Instanz einer Klasse**. Also nicht verwirren lassen, es gibt Klassen und Objekte/Instanzen.

Klassen und Objekte

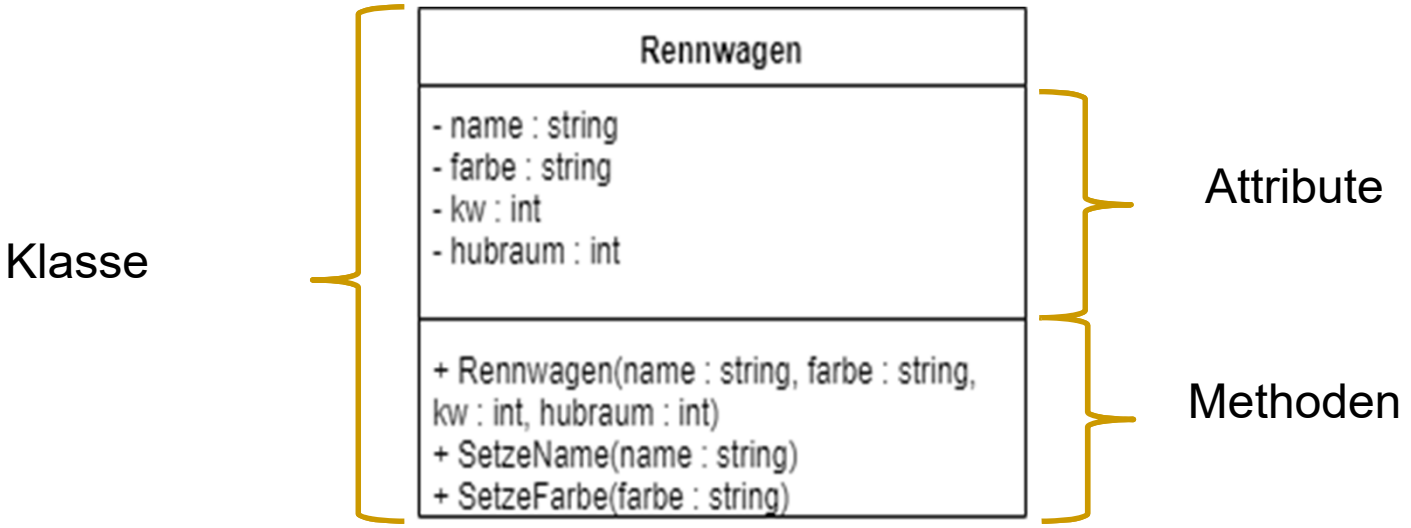


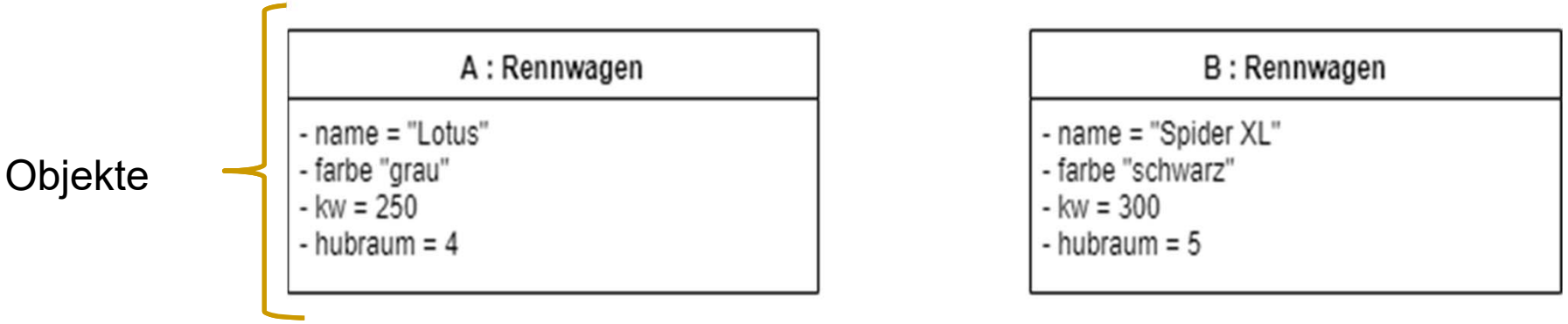
Name:	Lotus
Farbe:	blau
KW:	250
Hubraum:	4 Liter



Name:	Spider XL
Farbe:	schwarz
KW:	300
Hubraum:	5 Liter

Klasse und Objekte





Zusammenfassung von Klassen und Objekten

- Wir brauchen Klassen (Baupläne) um Objekte (Gegenstände) zu erzeugen
- Mit Klassen selbst können wir „nichts“ machen, nur mit Objekten
- Von einer Klasse können wir beliebig viele Objekte erstellen
- Objekte werden auch als Instanzen bezeichnet

Warum das Ganze?

- Realitätsnäher aber kompliziert
- Aber Klassen modularisieren eine Anwendung in unabhängige Einheiten.
- *Wiederverwendbarkeit der einzelnen Stücke (Module)*
- *Wartbarkeit:* Eine Klasse kann als eigene, separate und unabhängige Einheit getestet und geändert werden.

Aufbau einer Klasse

```
class Haus
{
   //Attribute
   public int länge;
   public int breite;
   private string ort;
   //Methoden
   public int Berechne()
   {
      int erg = länge*breite;
      return erg;
   }
}
```

Zugriffs-modifizierer	Beschreibung
`public`	Der Zugriff unterliegt keinerlei Einschränkungen.
`private`	Der Zugriff auf ein als `private` definiertes Mitglied ist nur innerhalb der Klasse möglich, die den Member definiert. Alle anderen Klassen sehen private Member nicht. Deshalb ist darauf auch kein Zugriff möglich.
`protected`	Der Zugriff auf `protected` Member ähnelt dem Zugriff auf als `private` definierte Member. Die Sichtbarkeit ist in gleicher Weise eingeschränkt, jedoch sind als `protected` definierte Mitglieder in abgeleiteten Klassen sichtbar.
`internal`	Der Zugriff auf `internal` Member ist nur aus den Klassen heraus möglich, die sich in derselben Anwendung befinden.
`protected internal`	Stellt eine Kombination aus den beiden Modifizierern `protected` und `internal` dar.

Aufbau einer Klasse

```
class Haus
{
   //Attribute
   public int länge;
   public int breite;
   private string ort;
   //Methoden
   public int Berechne()
   {
      int erg = länge*breite;
      return erg;
   }
}
```

Zugriff auf Klassen

Beispiel Zugriff class Haus aus class Program

```
class Program
{
   static void Main(string[] args)
   {
      Haus haus1;
      haus1 = new Haus();
      haus1.länge = 20;   <- klappt!
      haus1.ort = „FB“;   <- klappt nicht!
      haus1.Berechne();   <- klappt (gibt
   }                      einen Wert zurück)
}
```

Konstruktoren

- Spezielle Methode einer Klasse, die nur mithilfe des `new` – Operators aufgerufen werden kann.
- **Beim Erstellen** eines Objektes bzw. einer Instanz **wird immer der Konstruktor** der jeweiligen Klasse oder Stuktur **aufgerufen.**
- Auch wenn **nicht** extra ein Konstruktor geschrieben wurde, existiert dieser doch.

Konstruktor

- Konstruktoren heißen wie der Klassenname.
- Konstruktoren haben keinen Rückgabewert, auch nicht void.
- Eine Klasse oder Struktur kann über mehrere Konstruktoren mit jeweils unterschiedlichen Übergabewerten verfügen.
- Konstruktoren erzeugen ein Objekt der Klasse. Werden keine Parameter übergeben, so werden die Attributwerte mit Nullwerten initialisiert.

Konstruktor

```
public class Auto
{
   int ps;
   int maxV;
   public Auto() // Hier beginnt der Konstruktor
   {
      ps   = 160; // Initialisierungen durchführen ...
      maxV = 220;
   } // Hier endet der Konstruktor
}
```

Mitgeben von Parametern

```
Auto a1 = new Auto(160,220); //neue Instanz mit
                              Eigenschaften ps=160,
                              maxV=220 erzeugen
public class Auto
{
   int ps;
   int maxV;
   public Auto(int ps, int maxV)
   {
      this.ps   = ps; //Initialisierungsparameter übernehmen
      this.maxV = maxV;
   }
}
```

Mitgeben von Parame

```
Auto a1 = new Auto(160,220); //

public class Auto
{
   int ps;
   int maxV;
   public Auto(int ps, int maxV)
   {
      this.ps   = ps; //Initialisierungsparameter übernehmen
      this.maxV = maxV;
   }
}
```

Was macht der Befehl this? Dieser Kennzeichnet das Akutelle Objekt einer Klasse. Man bekommt so Zugriff auf Member, die durch gleiche Variablenbezeichnung überlagert wurden.

Überladen von Konstruktoren

→ Aufruf: Auto a1 = new Auto(); //Konstruktor ohne Parameter

→ Aufruf: Auto a1 = new Auto(160,220); //Konstruktor mit Parameter

```
public class Auto
{
   int ps;
   int maxV;
   public Auto(){} //ohne Parameter
   public Auto(int ps, int maxV) //mit Parameter
   {
      this.ps   = ps; //Initialisierungsparameter übernehmen
      this.maxV = maxV;
   }
}
```

static, was?

Static Variablen oder Methoden gehören zur Klasse (Typ) nicht zum erzeugten Objekt!!!!
Sie lassen sich somit nicht über das Objekt ansprechen.
Hier wird die static Variable autoAnzahl, als Zähler verwendet, welcher die erzeugten Objekte speichert.

```
public class Auto
{
   int ps;
   int maxV;
   static int autoAnzahl = 0; //Startwert
   public Auto(int ps, int maxV) //mit Parameter
   {
     autoAnzahl++;   //
      this.ps   = ps; //Initialisierungsparameter übernehmen
      this.maxV = maxV;
   }
}
```

Destruktor

- Genau wie einen Konstruktor gibt es auch einen Destruktor, dieser ist dafür verantwortlich das von einem Objekt beanspruchte Ressourcen wieder freigegeben werden.
- In der Regel wird dieser jedoch nicht selbst geschrieben.
- Der Programmierer kann auch nicht steuern, wann der Destruktor aufgerufen wird, da dies durch den Garbage Collector bestimmt wird.

Aufgaben

1. Schreibe eine Klasse `Fachbuch`, `Freizeitbuch` und `Film` und überlege dir für diese Klassen jeweils mind. 3 Eigenschaften
   a) Schreibe für die Klasse `Fachbuch`, `Freizeitbuch` und `Film` jeweils einen Konstruktor mit dem auch Argumente übergeben werden.
   b) Schreibe mindestens zu einer der Klassen einen weiteren Konstruktor mit anderer Werteübergabe (Bsp. nur einen Teil der Attribute eines Objekts mit Werten belegen).
   c) Schreibe die entsprechenden Objektaufrufe in das Hauptprogramm. (Erzeuge ein Objekt der jeweiligen Klasse im Hauptprogramm, mithilfe der Konstruktoren)

Aufgaben

2. Schreibe eine Klasse `BodyMassIndex` mit den Eigenschaften `groesse` und `gewicht` vom Typ `double`.
   - Die Klasse soll weiterhin über eine Methode zur Berechnung des BMI verfügen.
3. Schreibe eine Klasse `Geometrie`, welche die Flächen und Umfänge der Formen `Rechteck`, `Kreis` und `Dreieck` berechnet.
   - Umsetzung der Aufgabe soll mit `Rechteck`, `Kreis` und `Dreieck` als Klasse erfolgen
   - Nicht vergessen testet alle eure Klassen und Funktionen!!!

Aufgaben

Schreibe folgende Klassen:

- die Klasse `Kunde`. Jeder `Kunde` verfügt über eine eindeutige `kundenNr`, einen `namen` und ein `konto`.
- die Klasse `Konto`. Jedes `Konto` hat eine eindeutige `kontoNr`, einen `kontostand` und einen `inhaber` vom Typ `Kunde`. Folgende Transaktionen sollen mit einem Konto durchfgeführt werden können. Geld einzahlen und abheben, sofern das Konto noch nicht leer ist.
- Erstelle zwei Kunden mit je einem Konto und teste deine Methoden auf Fehler.