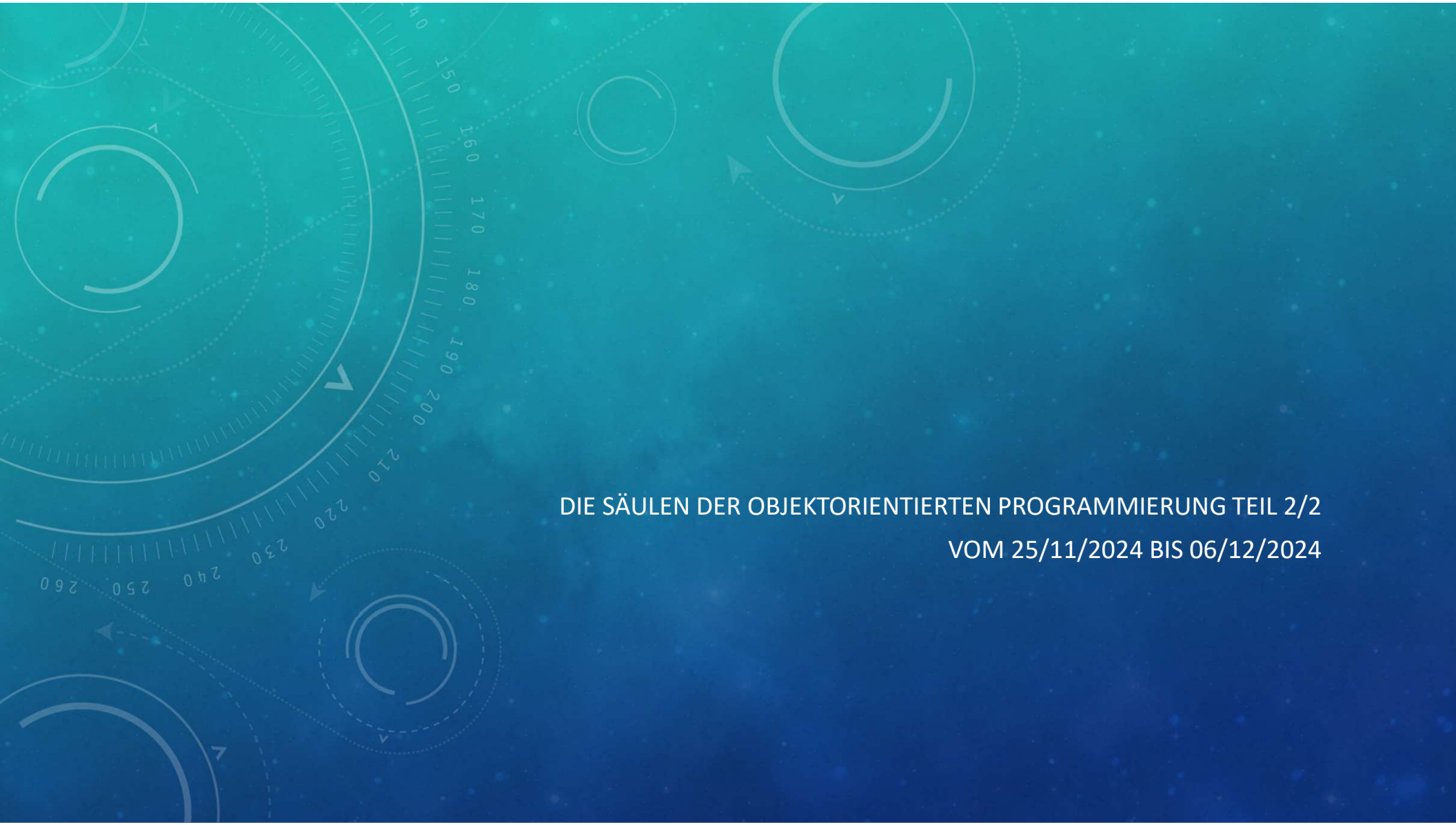


DIE SÄULEN DER OBJEKTORIENTIERTEN PROGRAMMIERUNG TEIL 2/2

VOM 25/11/2024 BIS 06/12/2024

OOP DEFINITION

Die **objektorientierte Programmierung** (kurz **OOP**) ist ein auf dem Konzept der Objektorientierung basierendes Programmierparadigma, welches Flexibilität und Wiederverwendbarkeit von Programmen fördert.

Die Grundidee der objektorientierten Programmierung ist, Daten und Funktionen, die auf diese Daten angewandt werden können, möglichst eng in einem sogenannten *Objekt* zusammenzufassen und nach außen hin zu *kapseln*, so dass Methoden fremder Objekte diese Daten nicht versehentlich manipulieren können.

Im Gegensatz dazu beschreibt das vor der OOP vorherrschende Paradigma eine strikte Trennung von Funktionen (Programmcode) und Daten, dafür aber eine schwächere Strukturierung der Daten selbst.

WARUM OBJEKTORIENTIERT PROGRAMMIEREN?

- Software lässt sich schneller entwickeln
- ist leichter zu warten
- weniger Fehleranfällig sowie einfacher zu debuggen
- Source Code ist übersichtlicher
- Wiederverwendbar (Frameworks)

WAS ZEICHNET OBJEKTORIENTIERTE PROGRAMMIERUNG AUS?

- *Abstraktion*
  Ein Objekt verkörpert einen Arbeiter der verschiedene Eigenschaften hat und verschiedene Aufgaben übernehmen kann.

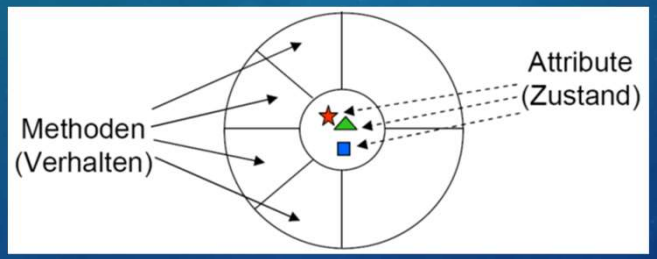
- *Kapselung*
  Jedes Objekt ist einer Kapsel, das heißt Eigenschaften und Verhaltens-weisen die in dem Objekt implementiert werden überschreiben nicht etwaig gleichlautende Eigenschaften außerhalb des Objektes.

- *Vererbung*
  Es können allgemeine Verhaltensweisen und Eigenschaften von Objekten in einen Oberklasse heraus generalisiert werden. So kann doppelter Code auf leichte Weise vermieden werden.

- *Polymorphie*
  Klassen die von einer Oberklasse allgemeine Eigenschaften und Verhaltens-weisen geerbt hat können diese überschreiben um so ein spezialisiertes Verhalten zu erzeugen.

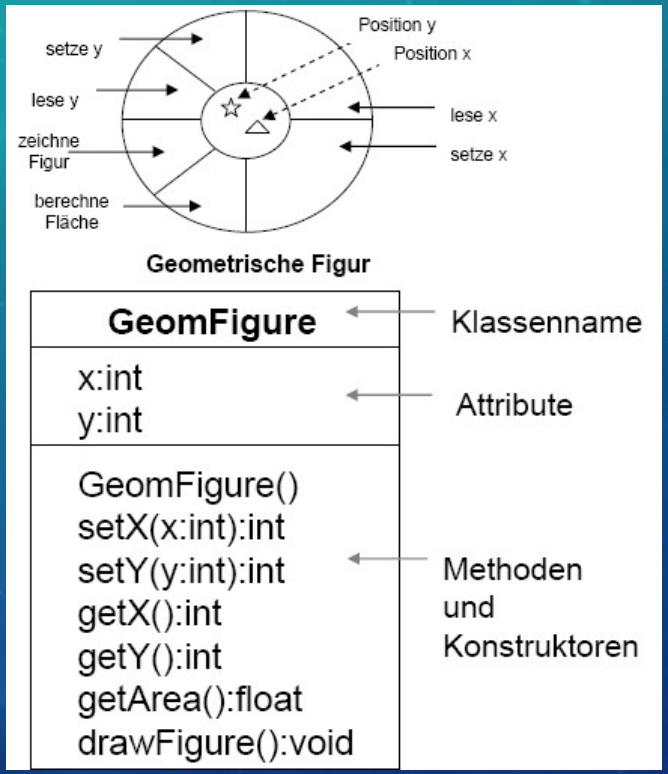
OOP MIT JAVA: OBJEKTE

- (Software-)Objekte besitzen einen internen **Zustand** und haben ein festgelegtes **Verhalten**
- Ein Objekt
  - verwaltet seinen Zustand durch Variablen, Eigenschaften oder **Attribute** genannt
  - implementiert sein Verhalten durch Funktionen, **Methoden** genannt
- Ein Java Objekt kann zum Beispiel eine Person, ein Ball oder ein Raumschiff sein.



UML NOTATION

- Die Unified Modeling Language (UML) ist eine Notation zur Spezifikation und Visualisierung von Modellen für Softwaresysteme
- In den folgenden Folien werden **Klassen-diagramme** zur Beschreibung von Klassen und deren Beziehungen untereinander verwendet



KLASSEN UND OBJEKTE (INSTANZEN)

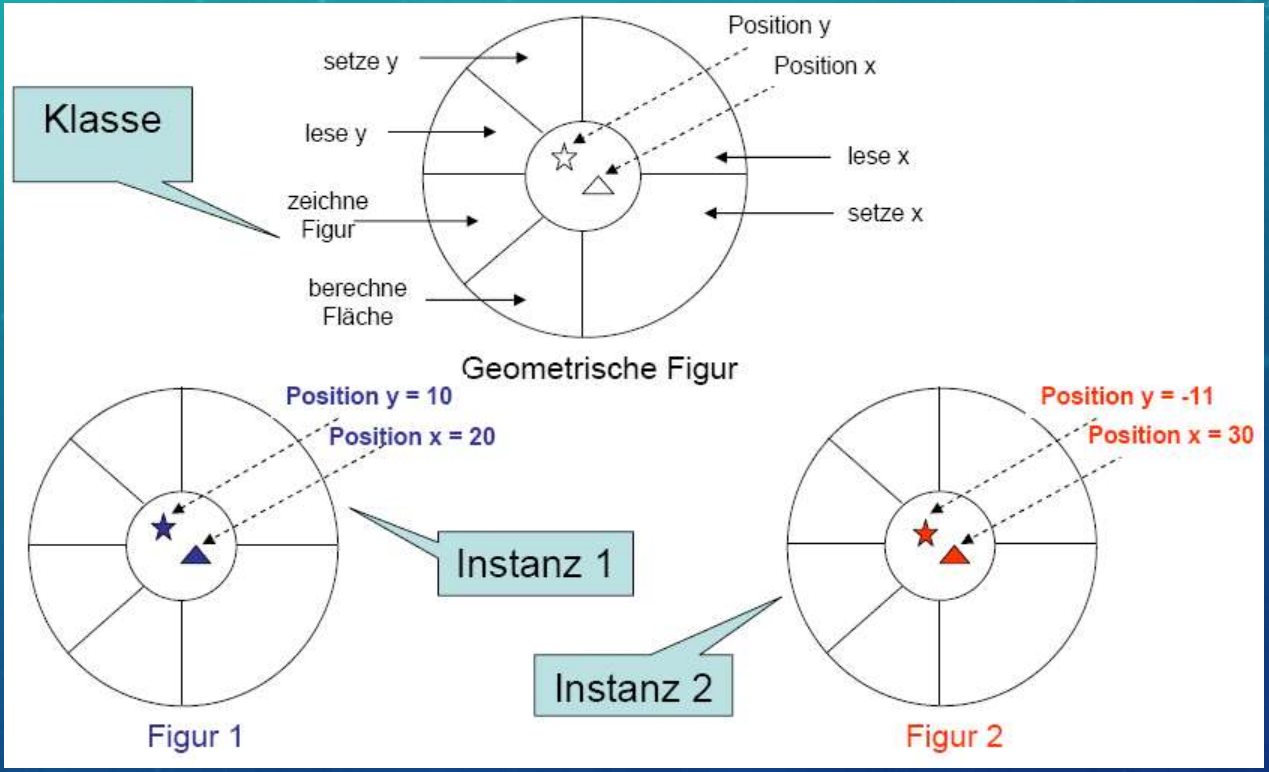
Klassen sind Vorlagen für die Erzeugung von Objekten

- Jedes Objekt gehört einer Klasse an.
- Die Attribute und Methoden der Objekte werden in der zugehörigen Klasse definiert
- Alle Objekte einer Klasse besitzen dasselbe Verhalten, da sie dieselbe Implementierung der Methoden besitzen
- Objekte einer Klasse werden auch als **Instanzen** (*instance*) oder **Exemplare** bezeichnet
- Klassen können auch Methoden und Attribute besitzen, die für die Klasse selbst und nicht für jede Instanz gelten. Diese werden als **Klassen-Methoden/-Attribute** bezeichnet (im Gegensatz zu **Instanz-Methoden/-Attribute**)

ERZEUGUNG VON OBJEKTEN

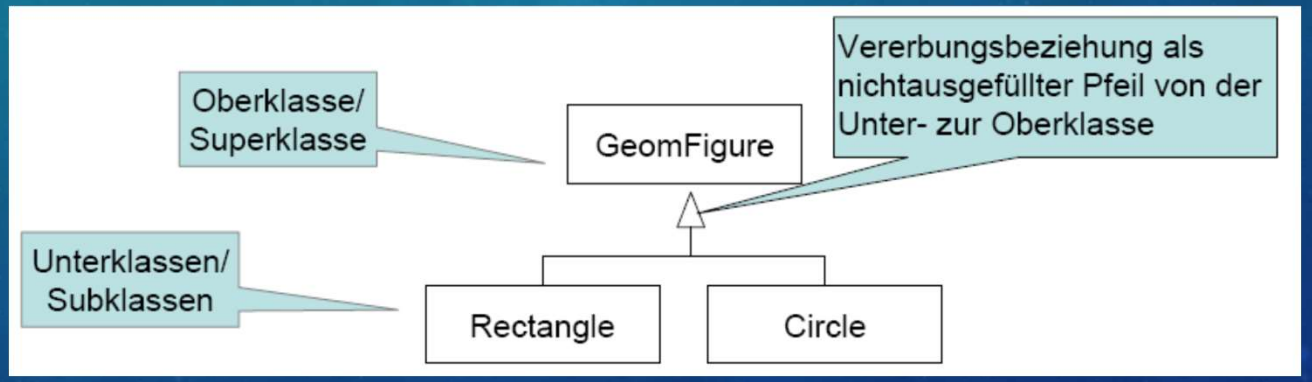
- Es wird Speicherplatz für das Objekt auf dem Heap allokiert (abhängig von den Datentypen der Attribute)
- Bei der Erzeugung wird ein **Konstruktor** (spezielle Methode) ausgeführt, der die Initialisierung des Objekts vornimmt
- Es werden den Attributen Werte zugeordnet, wodurch der Zustand des Objekts definiert wird
  - entweder definiert durch den Konstruktor
  - oder Default-Werte (0 für Zahlen, *null* für Referenzen, etc.)
- Die Erzeugung eines Objekts erfolgt in den meisten Programmiersprachen mit dem **new-Operator**

BEISPIEL: EINE KLASSE UND ZWEI INSTANZEN (OBJEKTE) DER KLASSE

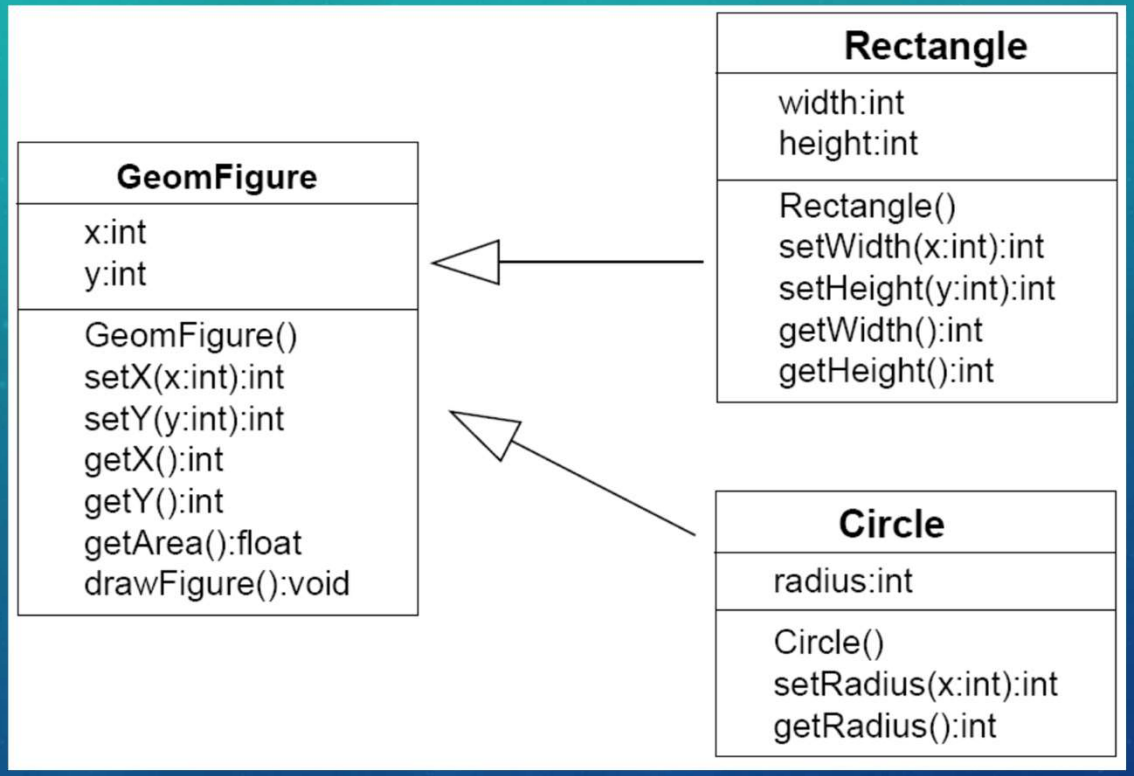


VERERBUNG

- Vererbung (*inheritance*) ist ein Konzept, wodurch Attribute und Methoden der Oberklasse auch den Unterklassen zugänglich gemacht werden
- Vererbungsbeziehungen zwischen Klassen stellen Generalisierungs- bzw. Spezialisierungsbeziehungen dar



VERERBUNG


GeomFigure
x:int
y:int
GeomFigure()
setX(x:int):int
setY(y:int):int
getX():int
getY():int
getArea():float
drawFigure():void
Rectangle
width:int
height:int
Rectangle()
setWidth(x:int):int
setHeight(y:int):int
getWidth():int
getHeight():int
Circle
radius:int
Circle()
setRadius(x:int):int
getRadius():int

VERERBUNG: DIE KLASSE *SQUARE*

- Die Klasse Square soll nun durch Vererbung von der Klasse GeomFigure abgeleitet werden.
- Wir benötigen einen Konstruktor, der neben der x- und y-Koordinate die Seitenlänge des Quadrates erhält.
- Weiterhin solle eine Methode zur Berechnung des Flächeninhaltes implementiert werden sowie die toString-Methode überschrieben werden.

ÜBERSCHREIBEN VON METHODEN

- Hat eine Methode einer Subklasse **die gleiche Signatur** einer Methode der Superklasse, so wird die Superklassenmethode überschrieben (**overriding**)
- GeomFigure definiert eine Methode toString():

```
String tmp="[" + myX + "," + myY + "]";
return tmp;
```

- Square überschreibt die Methode toString():

```
String tmp=super.toString();
tmp=tmp+" "+seitenL;
return tmp;
```

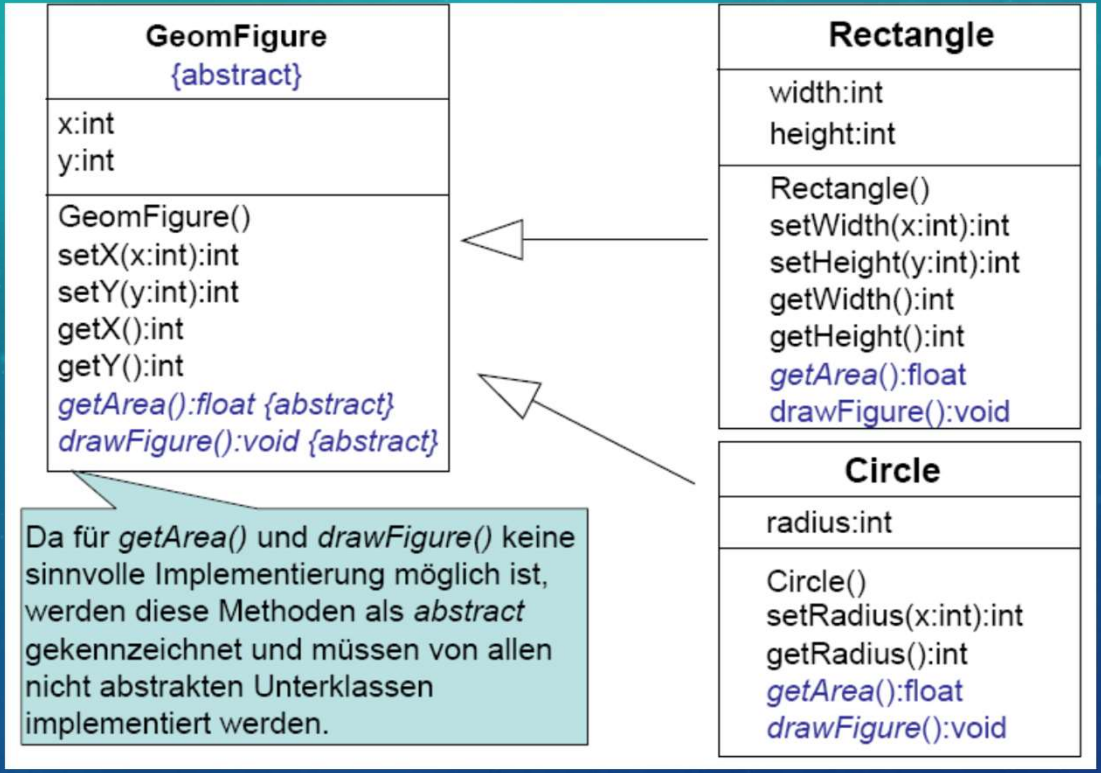
ÜBERLADEN VON METHODEN

- Gibt es mehrere Methoden mit demselben Namen aber mit **unterschiedlichen Signaturen**, so spricht man von Überladen (**overloading**).
- Beispiel:
  void println() ...
  void println (String s) ...
  void println (int i) ...

ABSTRAKTE KLASSEN

- Abstrakte Klassen sind Klassen, die mindestens eine **abstrakte Methode** haben
- Abstrakte Methoden besitzen keine Implementierung.
- Von abstrakten Klassen können **keine Instanzen** erzeugt werden
- Unterklassen müssen entweder die abstrakten Methoden implementieren oder sind ebenfalls abstrakt

BEISPIEL: ABSTRAKTE KLASSEN



ZUSAMMENFASSUNG

Objekte

- besitzen Attribute und Methoden
- gehören einer Klasse an

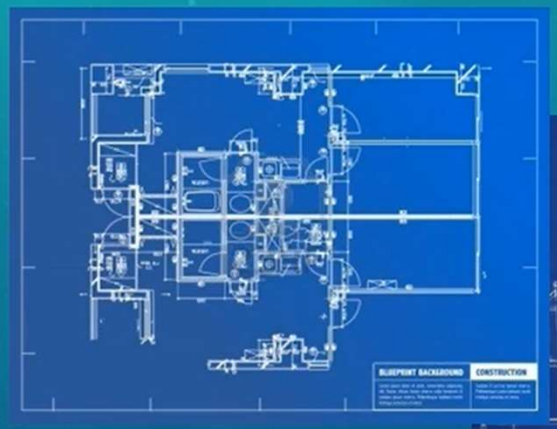
Klassen

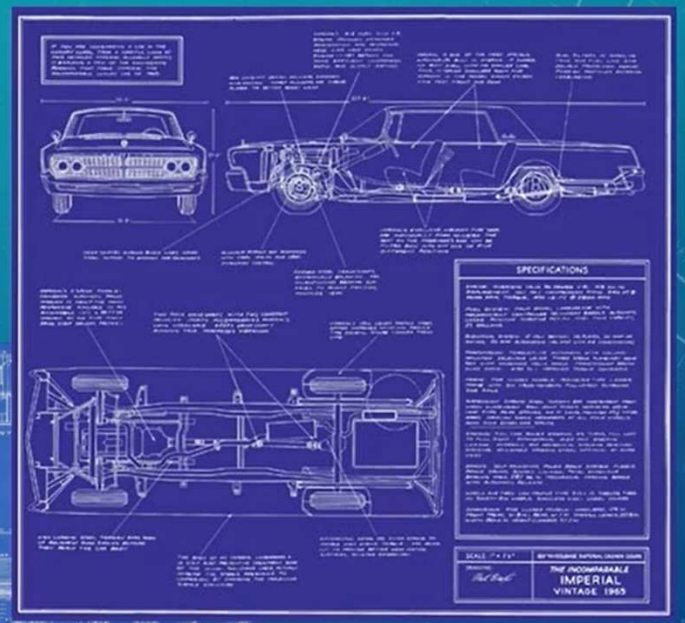
- legen Methoden und Attribute der Objekte fest (Bauplan)
- können Klassen-Methoden/-Attribute besitzen
- können abstrakt sein
- Objekte werden auf dem Heap erzeugt
- den Attributen werden Werte zugeordnet
- Konstruktoren werden ausgeführt
- Vererbung: Attribute und Methoden der Oberklasse werden den Unterklassen zugänglich gemacht

VORTEILE VON OOP

- Wiederverwendung von Code
- Einfachere Anpassbarkeit von Code, damit auch bessere Wartbarkeit
- Objektorientierte Systeme ermöglichen einen strukturierten Systementwurf
- Software Engineering: OOA, OOD, OOP
- Nachteile: Lernaufwand, bei einfachen Programmen evt. mehr Code

WAS SIND KLASSEN?

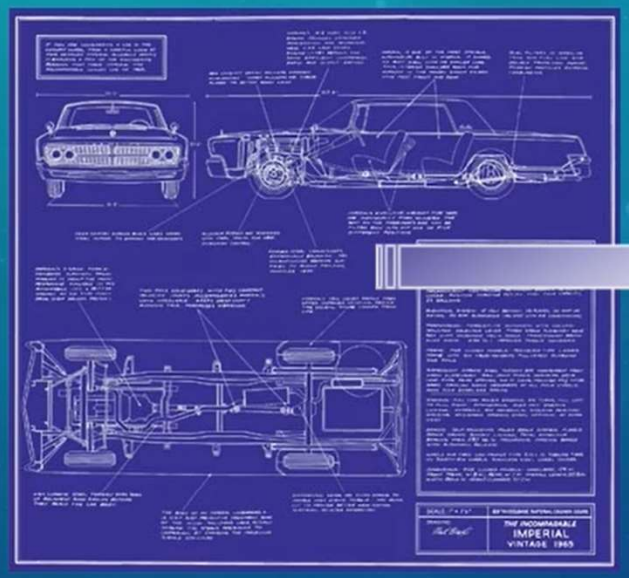




Bauanleitungen - Blueprints

allgemeine Anweisungen zum Erstellen von Objekten

OBJEKTE



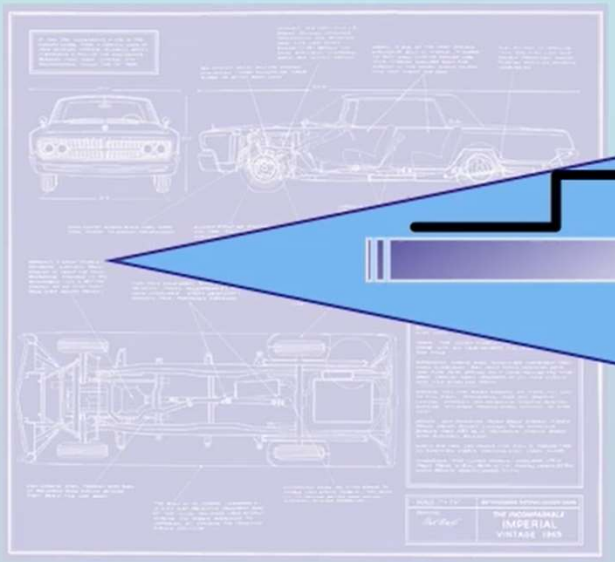
Blueprint [Klasse]

Zusammenbauen
[Initialisieren]

Exemplar [Objekt]

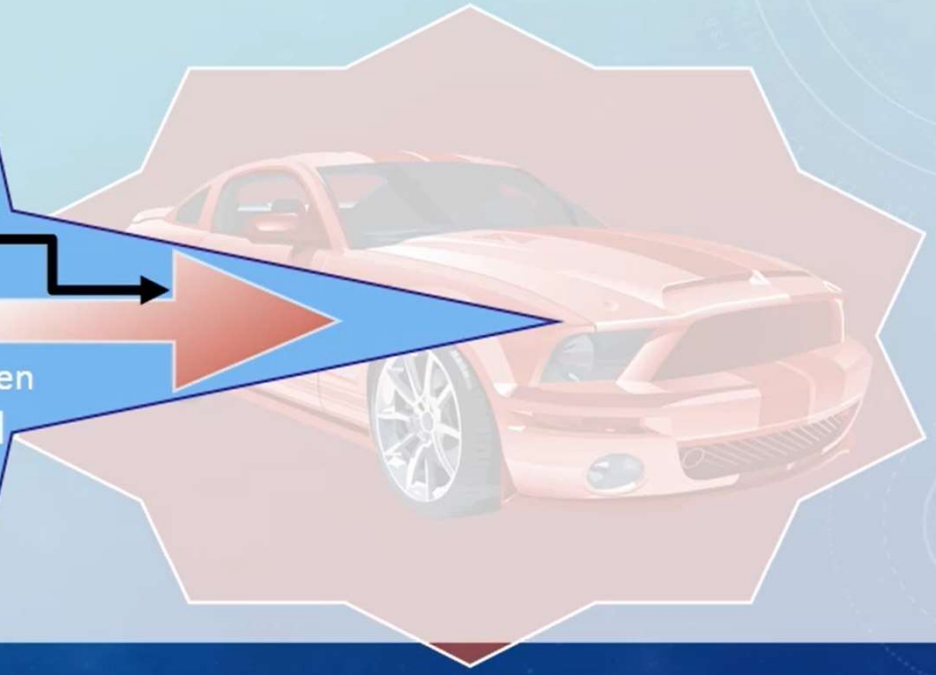
ATTRIBUTE

OBJEKTE



-ROT-

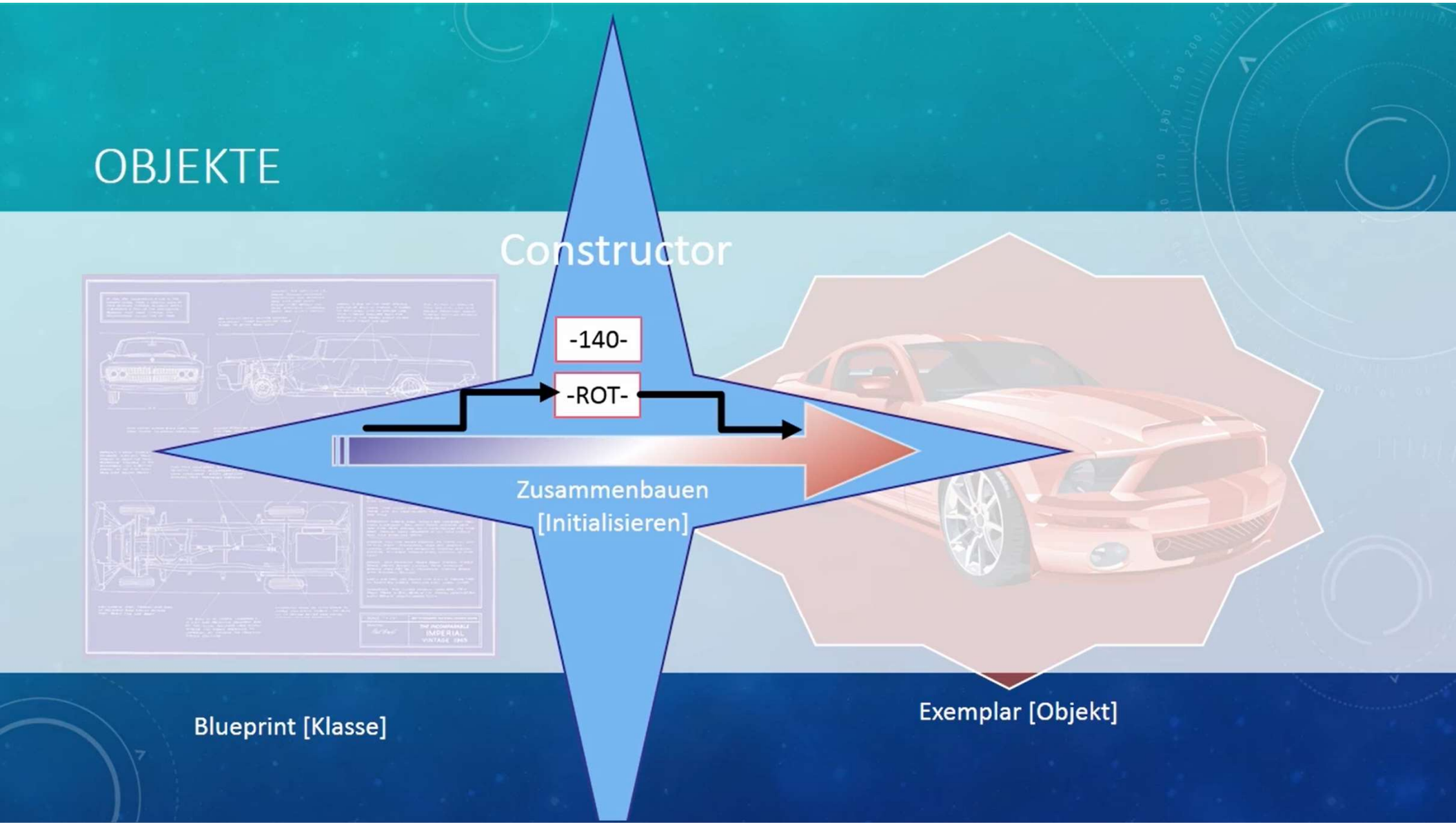
Zusammenbauen
[Initialisieren]



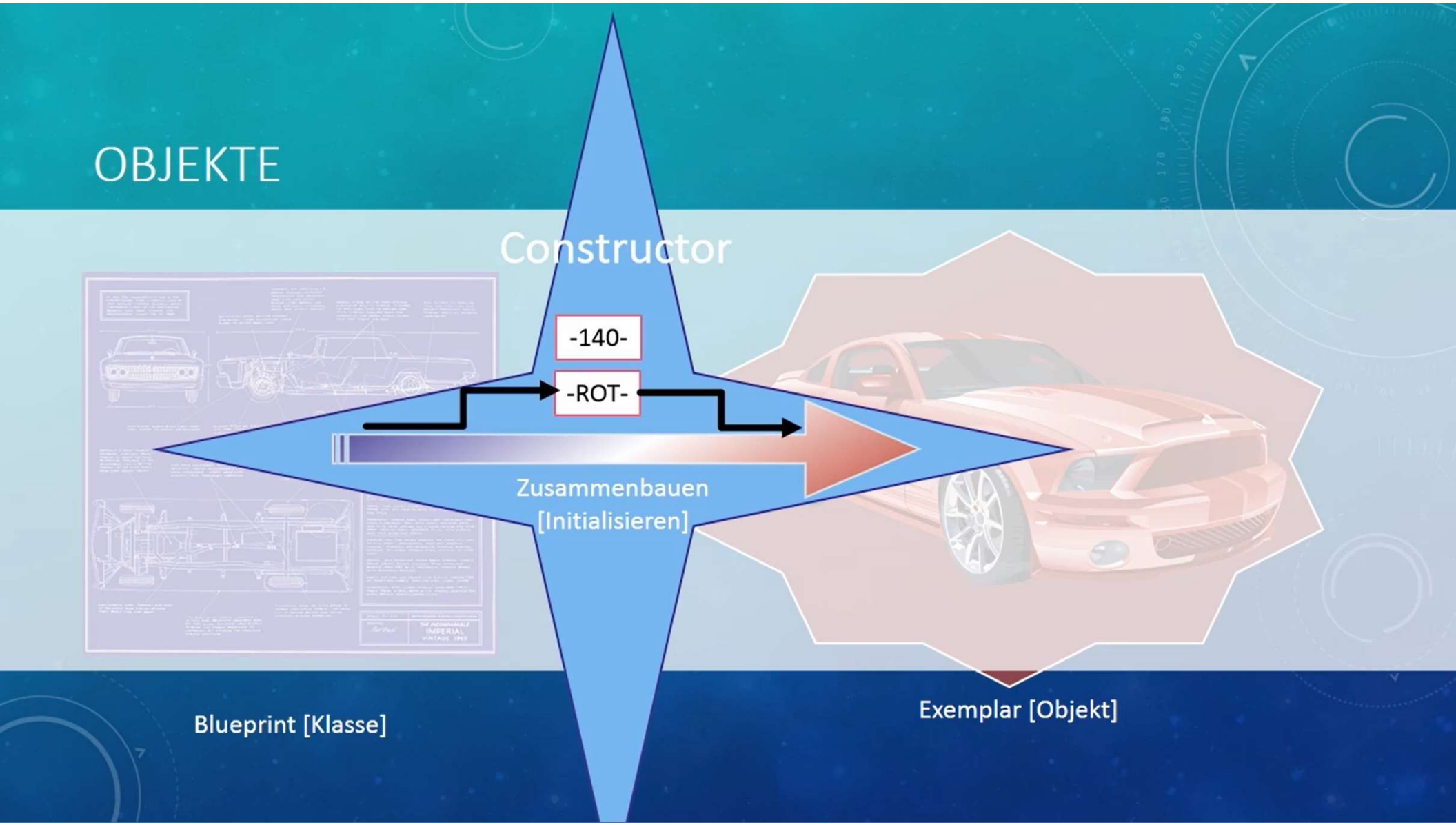
Blueprint [Klasse]

Exemplar [Objekt]

OBJEKTE


Constructor
-140-
-ROT-
Zusammenbauen
[Initialisieren]
Blueprint [Klasse]
Exemplar [Objekt]

OBJEKTE


Constructor
-140-
-ROT-
Zusammenbauen
[Initialisieren]
Blueprint [Klasse]
Exemplar [Objekt]

INITIALISIEREN /BAU VON OBJEKTEN

Festlegung von
Attributen für Auto:

Farbe

Maximalgeschwindigkeit


ATTRIBUTE
MPH
MILES

ATTRIBUTE

Objekt	Farbe	Maximalgeschwindigkeit
1	rot	120
2	gelb	110
3	grün	100
4	blau	110
5	lila	100

Attribute immer mit Werten füllen!

METHODEN

Attribute

Maximalgeschwindigkeit

Farbe

METHODEN

weitere Attribute

momentaneGeschwindigkeit (<120MPH)

Lenkradausrichtung

Aufgaben

Beschleunigen

Bremsen

Lenken

Attribute

Maximalgeschwindigkeit

Farbe

METHODEN

weitere Attribute

momentaneGeschwindigkeit (<120MPH)

Lenkradausrichtung

Aufgaben

Beschleunigen

Bremsen

Lenken

Attribute

Maximalgeschwindigkeit

Farbe

METHODEN

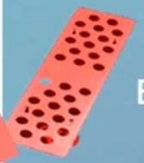
weitere Attribute

momentaneGeschwindigkeit (<120MPH)

Lenkradausrichtung

Aufgaben

Beschleunigen



Bremsen

Lenken

Attribute

Maximalgeschwindigkeit

Farbe

METHODEN

weitere Attribute

momentaneGeschwindigkeit (<120MPH)

Lenkradausrichtung

Aufgaben

Beschleunigen

Bremsen

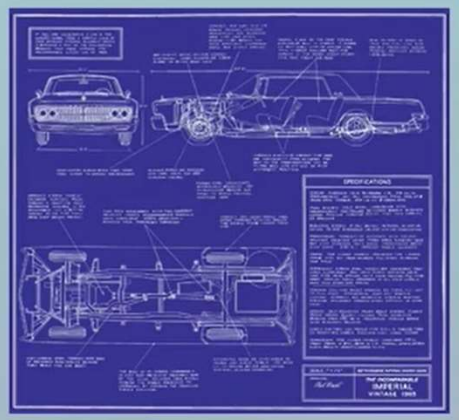
Lenken

Attribute

Maximalgeschwindigkeit

Farbe

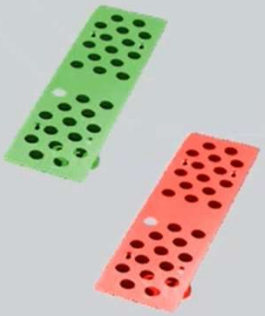
ZUSAMMENFASSUNG



Bauplan / Blueprint / Klasse

Konstruieren / Zusammenbauen / Initialisieren

Objekt / Exemplar



Aufgaben / Methoden