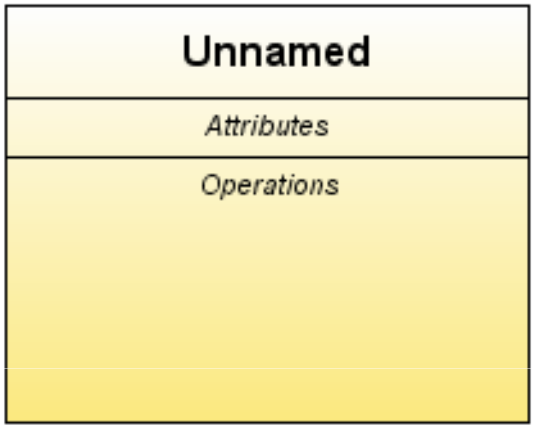
Grundlagen der Objektorientierung

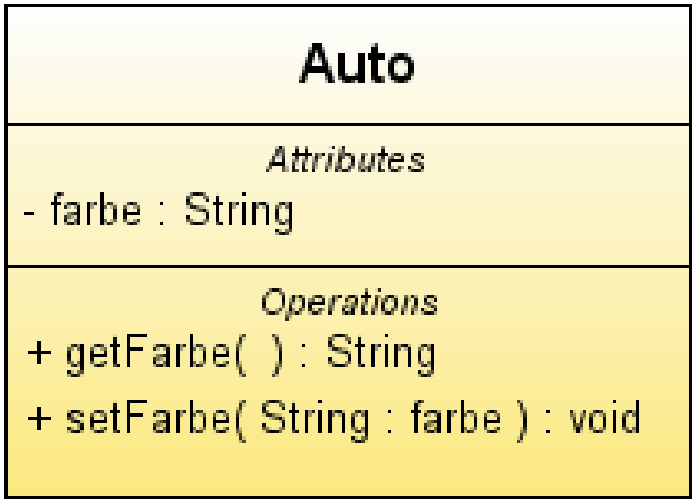
- Klassen, Attribute, Methoden
- Kapselung und Konstruktoren
- Vererbung
- Packages
- Interfaces und Adapterklassen



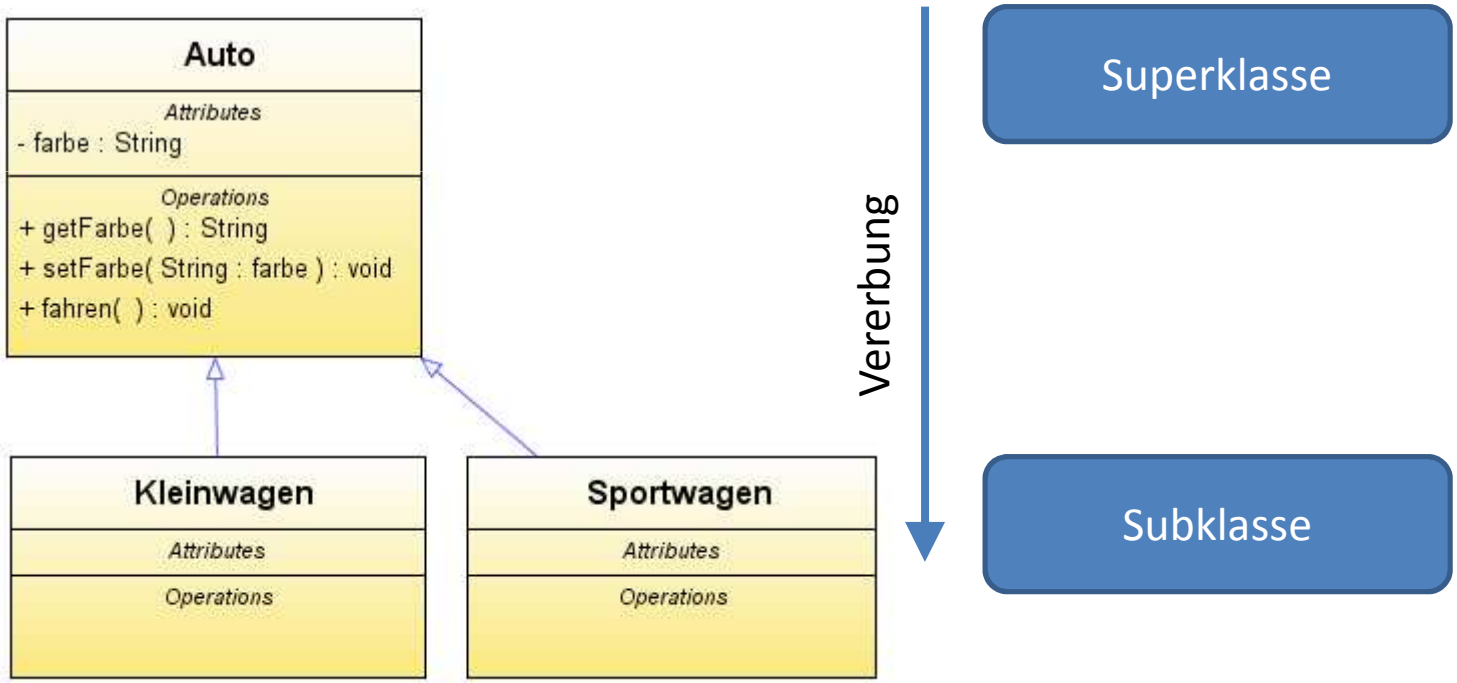
Klassen, Attribute, Methoden

```
package de.wieczorek.car;

/**
 * @author Marcel Wieczorek
 */
public class Auto {
          private String color;
          /**
           * @return the color
           */
          public String getColor() {
                    return color;
          }
          /**
           * @param color
           *             the color to set
           */
          public void setColor(String color) {
                    this.color = color;
          }
}
```



Vererbung


Auto
Attributes
- farbe : String
Operations
+ getFarbe( ) : String
+ setFarbe(String : farbe) : void
+ fahren( ) : void
Kleinwagen
Attributes
Operations
Sportwagen
Attributes
Operations
Vererbung
Superklasse
Subklasse

Vererbung

```java
package de.wieczorek.car;
/**
 * @author Marcel Wieczorek
 */
public class Auto {
	private String color;
	// Klassenvariable
	static String raeder = "4";
	/**
	 * @return the color
	 */
	public String getColor() {
		return color;
	}
	/**
	 * @param color
	 *            the color to set
	 */
	public void setColor(String color) {
		this.color = color;
	}
}
```

```java
package de.wieczorek.car;
/**
 * @author Marcel Wieczorek
 */
public class Sportwagen extends Auto {
}
```

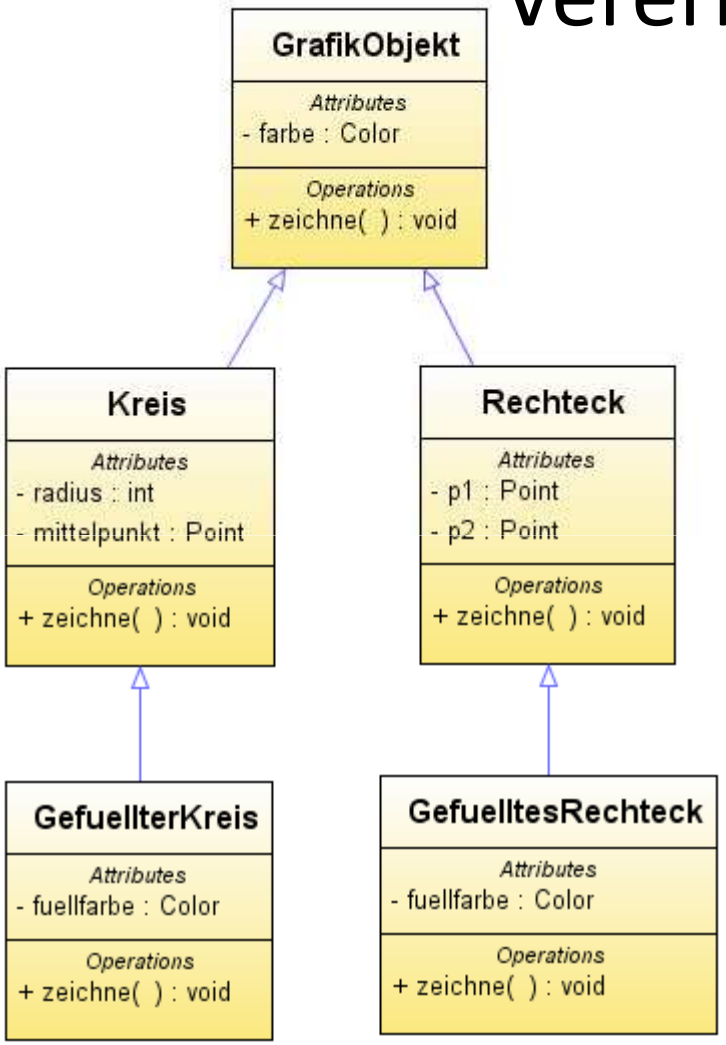
Vererbung

```
package de.wieczorek.car;
/**
 * @author Marcel Wieczorek
 */
public class Autohandel {
          public static void main(String[] args) {
                    String farbe = "grün";
                    String farbeSport = "rot";

                    Auto gruenesAuto = new Auto();
                    gruenesAuto.setColor(farbe);
                    System.out.print("Das Auto ist: ");
                    System.out.println(gruenesAuto.getColor());

                    Sportwagen ferrari = new Sportwagen();
                    ferrari.setColor(farbeSport);
                    System.out.print("Der Sportwagen ist: ");
                    System.out.println(ferrari.getColor());
          }
}
```

Vererbungsketten



- *Kreis* und *GefuellterKreis* erben **farbe**
- definieren eigene Methode **zeichne**
- fügen Attribute **radius** und **mittelpunkt** hinzu

Zuweisungskompatibilität

```
package de.wieczorek.graphics;
/**
 * @author Marcel Wieczorek
 */
public class TypeCast {
          /**
           * @param args
           */
          public static void main(String[] args) {
                    Rechteck firstForm = new GefuelltesRechteck();
                    Rechteck secondForm = new Rechteck();

                    GefuelltesRechteck firstFilled = (GefuelltesRechteck)
                    firstForm;
                    // Kein Compilerfehler, kein Laufzeitfehler

                    GefuelltesRechteck secondFilled = (GefuelltesRechteck)
                    secondForm;
                    // Kein Compilerfehler, aber Laufzeitfehler
          }
}
```

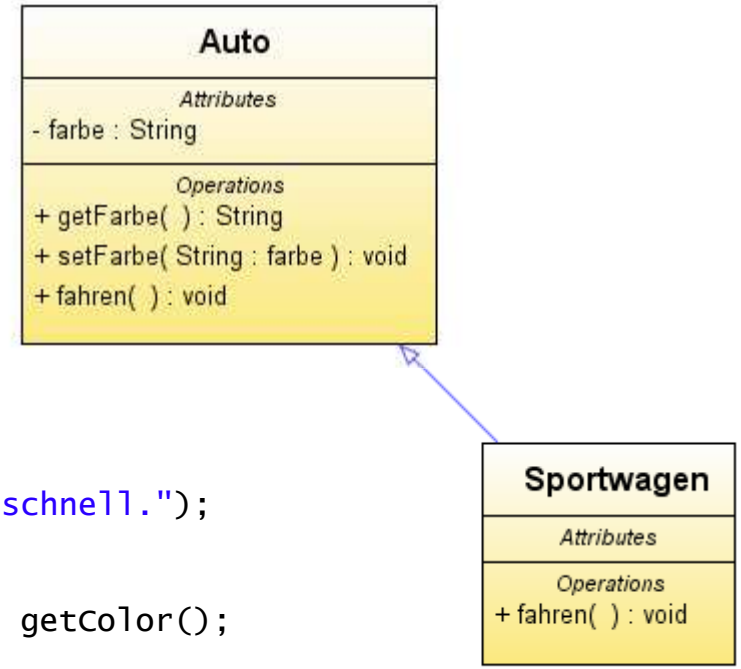
Überladen von Methoden

```
/**
 * @author Marcel Wieczorek
 */
public class Overload {
          public Overload() {
          }
          public Overload(String[] args) {
          }
          public void doSomething() {
          }
          public void doSomething(String[] args) {
          }
}
```

- Methoden oder Konstruktoren **vom Namen her gleich**
- Unterschiedliche Anzahl / unterschiedlicher Typ von Parametern

Überschreiben von Methoden

```
package de.wieczorek.car;
/**
 * @author Marcel Wieczorek
 */
public class Sportwagen extends Auto {
          public Sportwagen() {
                    this("rot");
          }
          public Sportwagen(String farbe) {
                    super(farbe);
          }
          public void fahren() {
                    super.fahren();
                    System.out.println("Ich fahre schnell.");
          }
          public String toString() {
                    return "Sportwagen, Farbe: " + getColor();
          }
}
```



- Methoden einer Subklasse haben **gleiche Signatur** wie die der Superklasse

Abstrakte Klassen

```
package de.wieczorek.shape;
/**
 * @author Marcel Wieczorek
 */
public abstract class Shape {
          private boolean filled = false;
          boolean isFilled() {
                    return this.filled;
          }
          abstract double getArea();
}
```

Abstrakte Klassen

```
package de.wieczorek.shape;
/**
 * @author Marcel Wieczorek
 */
public class Circle extends Shape {
          private double radius;
          double getRadius() {
                    return this.radius;
          }
          @Override
          double getArea() {
                    return Math.PI * (this.radius * this.radius);
          }
}
```

Interfaces

Syntax

```
[public] interface identifier
                    [extends interfaceName1 [, interfaceName2...]]
{
[public] [static] [final] type indentifier1 = value;
[public] type identifier2([parameter]);
}
```

Beispiel

```
package de.wieczorek.forms;
/**
 * Beschreibung eines Interfaces für zweidimensionale Formen
 *
 * @author Marcel Wieczorek
 */
public interface Forms2D {
          double getArea(); // Flächeninhalt
          double getPerimeter(); // Umfang
}
```

Interfaces

```
package de.wieczorek.forms;
import static java.lang.Math.PI;
/**
 * @author Marcel Wieczorek
 */
public class Circle implements Forms2D {
          private double radius;
          double getRadius() {
                    return this.radius;
          }

          // Implementierung der Interface-Methoden
          @Override
          public double getArea() {
                    return PI * this.radius * this.radius;
          }
          @Override
          public double getPerimeter() {
                    return 2D * PI * this.radius;
          }
}
```

Adapterklassen

```
package de.wieczorek.forms;
/**
 * @author Marcel Wieczorek
 */
public interface Forms2D2 extends Forms2D {
          // geerbte Methoden...
          void turn90(); // um 90 Grad drehen
}
```

```
package de.wieczorek.forms;
/**
 * @author Marcel Wieczorek
 */
public class Forms2D2Adapter implements Forms2D2 {
          @Override
          public void turn90() {

          }
          @Override
          public double getArea() {
                    return 0D;
          }
          @Override
          public double getPerimeter() {
                    return 0D;
          }
}
```

Adapterklassen

```
package de.wieczorek.forms;
import static java.lang.Math.PI;
/**
 * @author Marcel Wieczorek
 */
public class Circle2 extends Forms2D2Adapter {
          private double radius;
          double getRadius() {
                    return this.radius;
          }

          // Überschreiben der Adapter-Methoden
          @Override
          public double getArea() {
                    return PI * this.radius * this.radius;
          }
          @Override
          public double getPerimeter() {
                    return 2D * PI * this.radius;
          }
}
```