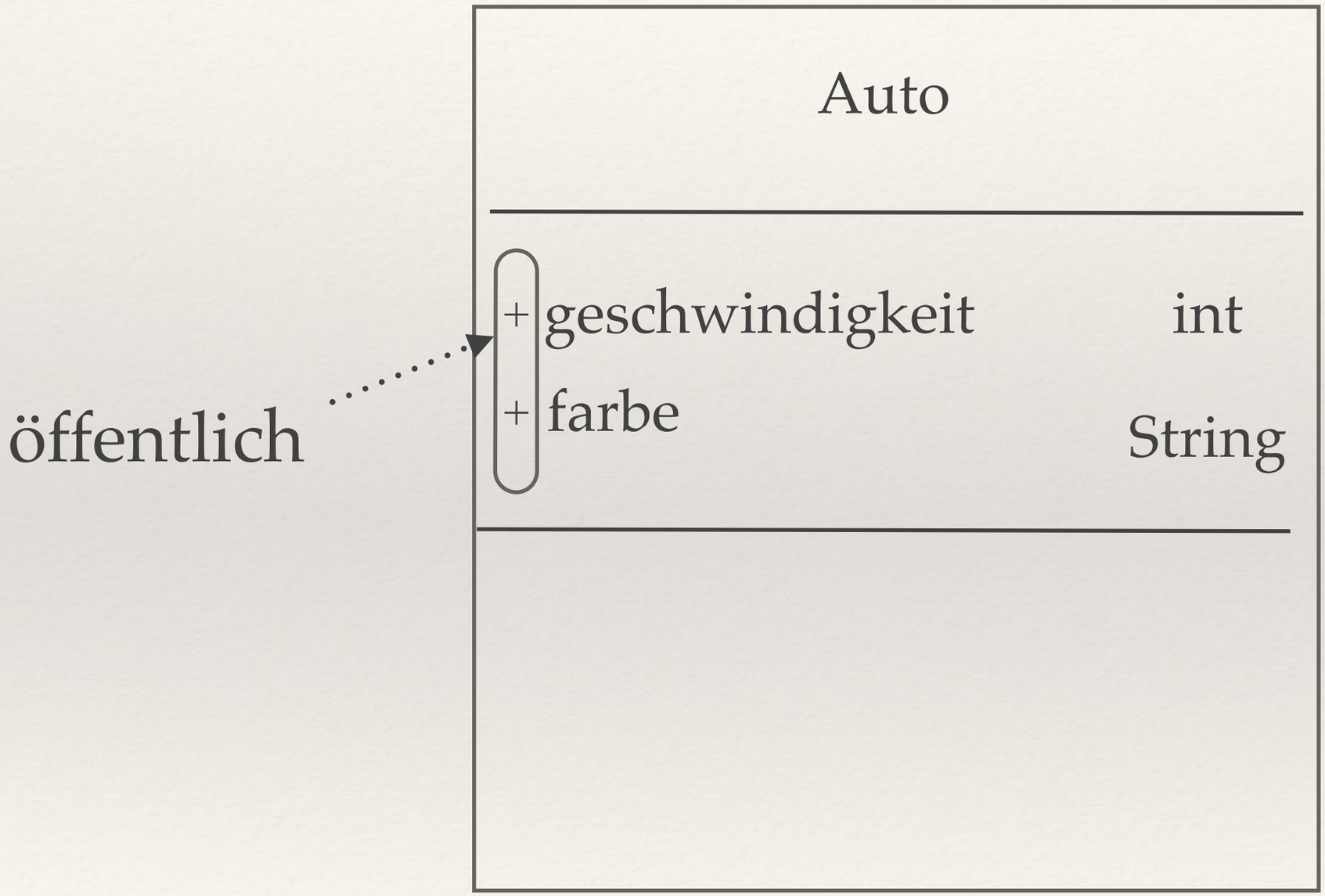
Umgang mit Klassen

Java Foundation Track by Carsten Bokeloh

Umgang mit Klassen

- Instanzmethoden
- Statische Komponenten
- Instantiierung & Initialisierung

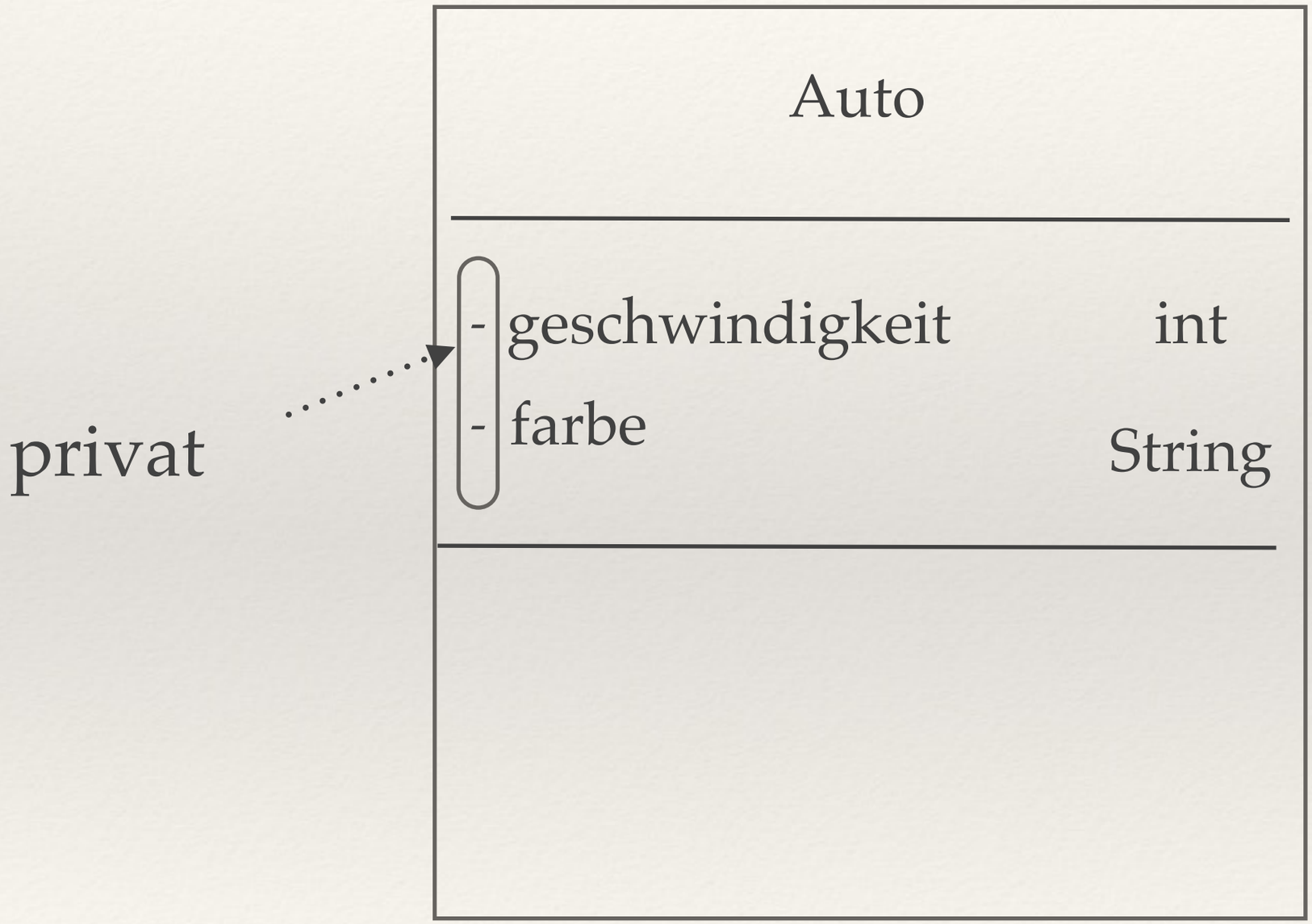
Die Klasse Auto



Zugriffsrechte

- data hiding
- ändern der Zugriffsrechte der Variablen

Die Klasse Auto

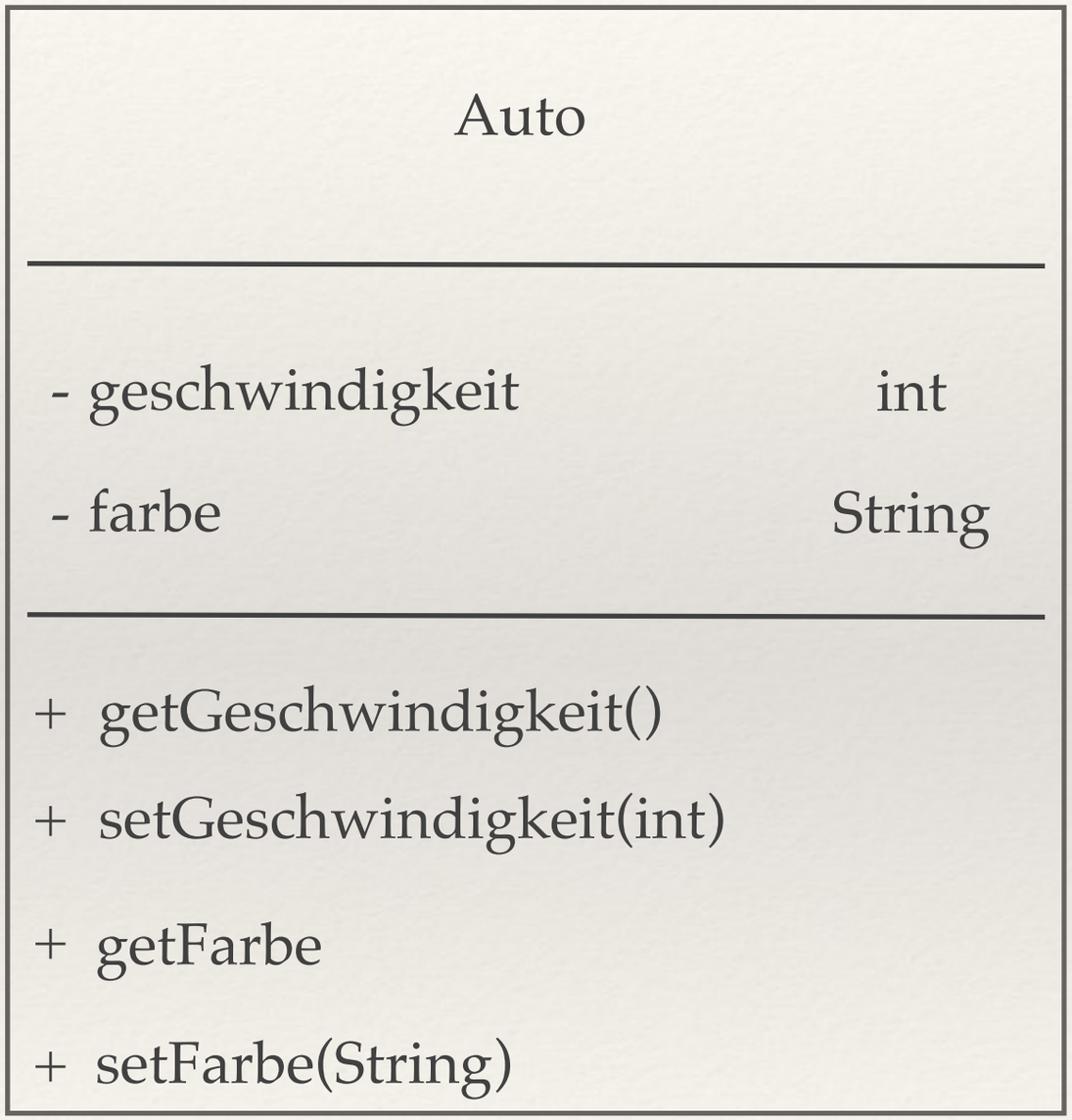


Instanzmethoden

```
public <<Rueckgabetyp>> <<Methodenname>> (<<Parameterliste>>)
{
    //Code
}
```

- ohne static hängt die Methode an einem konkreten Objekt
- hat damit Zugriff auf Ihre Instanzvariablen

Instanzmethoden



this

- Wie kann auf eine Objekt von innerhalb desselben Objekts zugegriffen werden ?
- „this“ ist eine Referenz auf das Objekt selbst
- Der Compiler verwende „this“ implizit vor Attributen und Methoden, denen kein Objekt oder Klassenname vorangestellt ist:

```
public int getAge() {
        return this.age;
}
```

this

- Zugriff auf überlagerte Variablen

```
public class Man {
    int age;
    public void setAge(int age) {
        this.age = age;
    }
    public int getAge() {
        return this.age;
    }

}
```

Instanzmethoden

- getter / setter
- Validierung
- zusätzliche Funktionalität

Statische Komponenten einer Klasse - Klassenvariablen

- private static <final>> <<Typ>> <<Variablenname>> = <<Initialwert>>
- in erster Linie als Konstanten verwendet, daher final
- Besipiel: Math.PI oder Integer.MAX_VALUE

Statische Komponeneten einer Klasse - Klassenmethoden

- public static <<TYP>> <<METHODENNAME>>
- Methoden, die nicht den Zustand eines Objektes nutzen
- Bsp: das schon bekannte Math.max(int a, int b)

Konstruktor

```
String marke;
String farbe;
int geschwindigkeit;
int baujahr;
```

```
public Auto() {

}
```

```
public Auto(String marke, String farbe, int geschwindigkeit, int baujahr) {
    this.marke = marke;
    this.farbe = farbe;
    this.geschwindigkeit = geschwindigkeit;
    this.baujahr = baujahr;
}
```

Konstruktor

Konstruktoren sind spezielle methodenähnliche Klassenstrukturen, die den Namen ihrer Klasse tragen und beim Erzeugen von Objekten der Klasse über das Schlüsselwort new aufgerufen werden.

überladene Konstruktoren

Auto.java

```
public class Auto {
        String marke;
        String farbe;
        int leistung;
        int baujahr;

        //1. Konstruktor
        public Auto() {
        }

        //2. Konstruktor
        public Auto(String marke, String farbe, int leistung, int baujahr) {
                this.marke = marke;
                this.farbe = farbe;
                this.leistung = leistung;
                this.baujahr = baujahr;
        }

        public void anlassen() {
                //Code zum Anlassen des Autos
        }

        public void beschleunigen () {
                //Code zum Beschleunigen des Autos
        }

        public void bremsen() {
                // Code zu Bremsen des Autos
        }
}
```

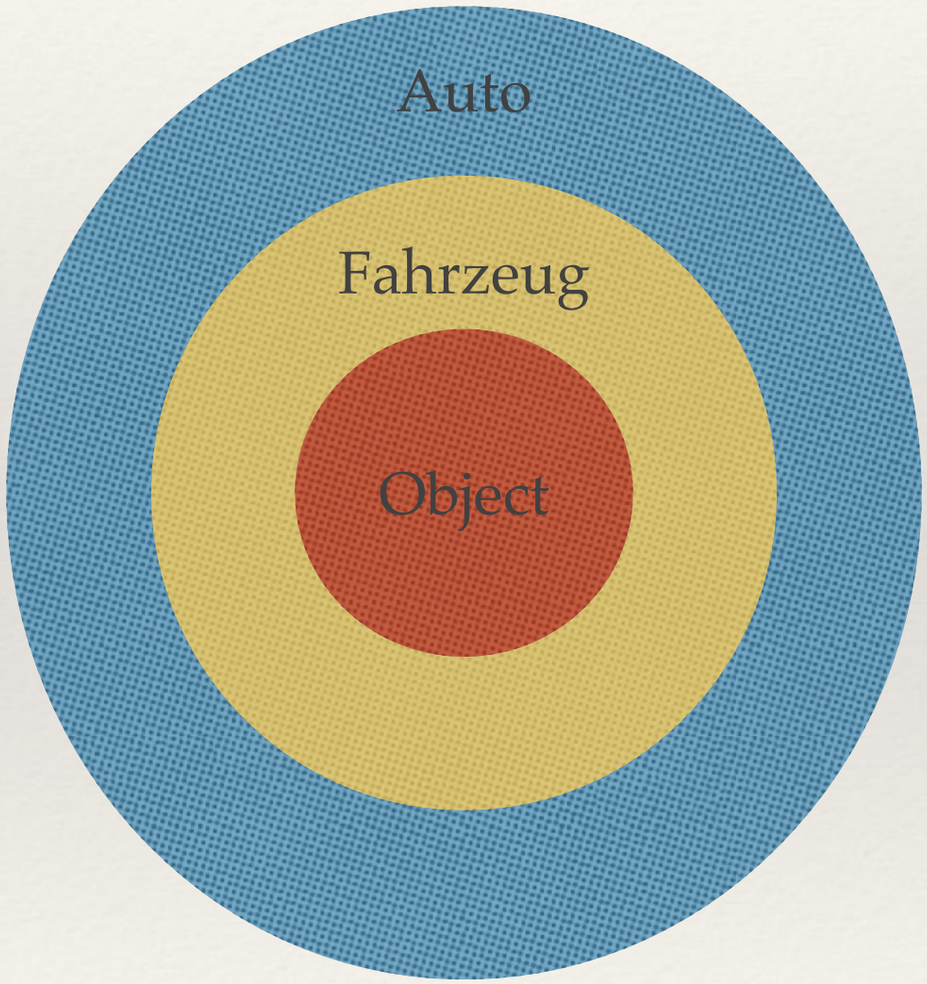
Verwendung des Konstruktors

- Auto bmw = new Auto();
- Auto vw = new Auto(“vw“, “schwarz“, 120, 2009);

Konstruktoren

- Der Konstruktor heißt immer so wie die Klasse
- Der Konstruktor verfügt über eine Parameterliste, in der wir Argumente vereinbaren können.

Konstruktorvererbung



Destruktor

```
public void finalize() {

}
```

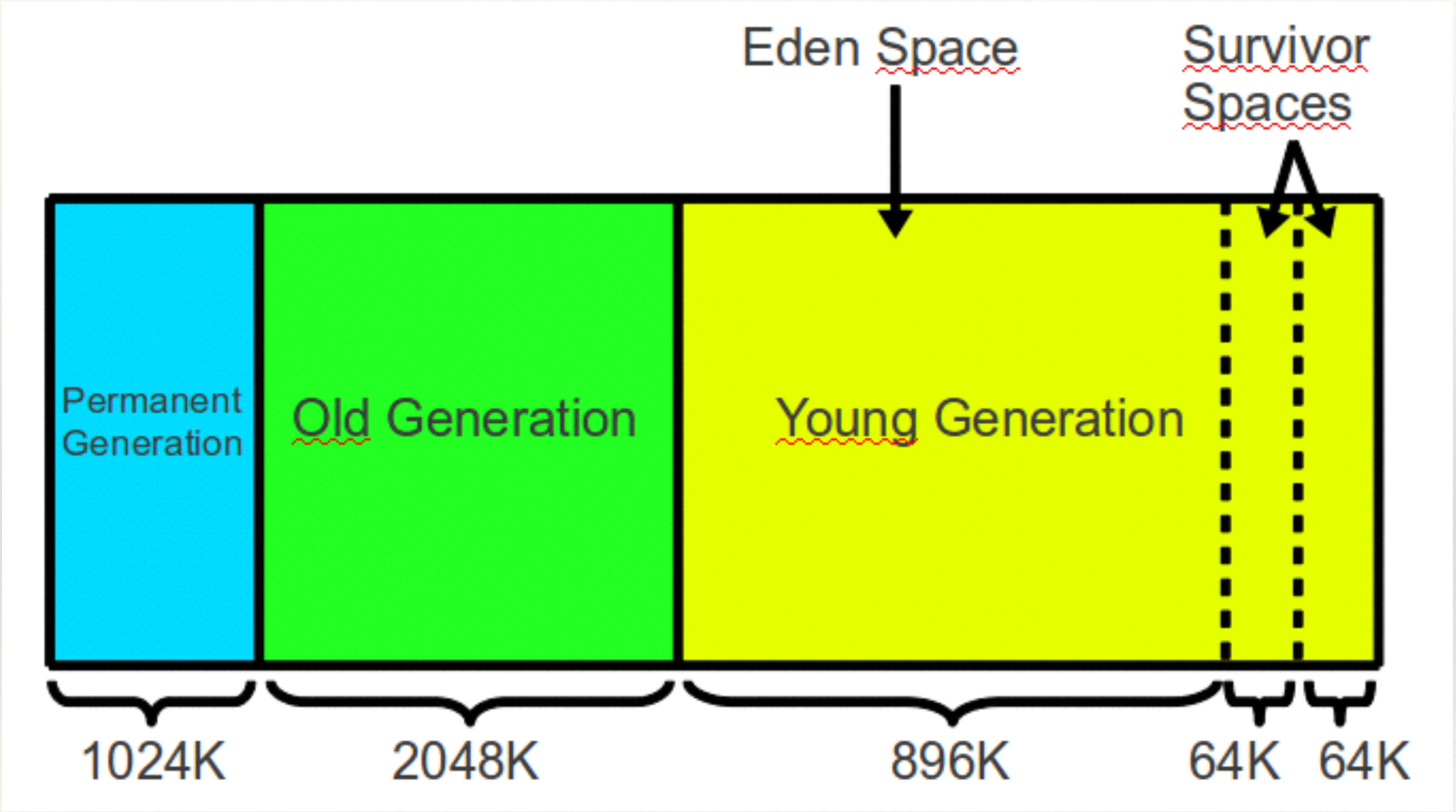
Destruktor

Die parameterlose Methode *finalize()* wird ausgeführt kurz bevor der *Garbage Collector* das Objekt zerstört und den vom Objekt belegten Speicher wieder freigibt.

Garbage Collector

- Ein mit niedriger Priorität laufender Thread
- Löscht alle Objekte, auf denen keine Referenz mehr besteht.
- Der Zeitpunkt, wann ein Objekt wirklich endgültig aus dem Speicher entfernt wird, ist unbestimmt.
- mit System.gc() kann die Ausführung erzwungen werden.

java heap


Eden Space
Survivor Spaces
Permanent Generation
Old Generation
Young Generation
1024K
2048K
896K
64K
64K

statische Initialisierer

```
static int i;
static float pi;
static String[] str;

static {
    i = 3;
    pi = 3.1415f;
    str = new String[]{"eins", "zwei"};
}
```

statische Initialisierer

- Statische Initialisierer haben nur Zugriff auf statische Komponenten einer Klasse
- Statische Initialisieret haben Zugriff auf alle (auch private) Teile einer Klasse
- Statische Initialisierer haben nur Zugriff auf statische Komponenten, die im Programmcode vor Ihnen definiert wurden

Im Buch gibt es mehr Infos unter Kapitel 5.1

http://openbook.galileocomputing.de/javainsel/javainsel_05_001.html#dodtp44e7828f-b21f-41d3-9c25-d36ace6e35b4

Übung

Gegeben ist folgende Klasse:

```
public class Reifen {

        private double druck;//Reifendruck

        public Reifen (double luftdruck) {
                druck = luftdruck;
        }
        public double aktuellerDruck () {
                return druck;
        }
}
```

Übung

Schreiben Sie eine Klasse Fahrzeug, die die Klasse Reifen verwendet und folgendes beinhaltet:
a.) private Instanzvariablen

- name vom Typ String(für die Bezeichnung des Fahrzeugs),
- anzahlReifen vom Typ int(für die Anzahl der Reifen des Fahrzeugs,
- reifenArt vom Typ Reifen (für die Angabe des Reifentypsdes Fahrzeugs) und
- faehrt vom Typ boolean(für die Information über den Fahrzustand des Fahrzeugs);

b.) einen Konstruktor, der mit den Parametern für Bezeichnung, Reifenanzahl und Reifendruck ausgestattet ist, in seinem Rumpf die entsprechenden Komponenten des Objekts belegt und außerdem das Fahrzeug in dn Zustand „fährt nicht“ versetzt.

c.) eine öffentliche Instanzmethode halteAn(), die die Variablefaehrt des Fahrzeug-Objektes auf false setzt;

d.)eine öffentliche Instanzmethode status(), die einen Informations-String über Bezeichnung, Fahrzustand, Reifenzahl und Reifendruck des Fahrzeug Objektes auf dem Bildschirm ausgibt.

Schreiben Sie ein Testprogramm, das in seiner main-Methode zunächst ein Fahrzeug(verwenden Sie Reifen mit 4.5 bar) und ein Auto (verwenedn Sie Reifen mit 1.9 bar) in Form von Objekten der Klasse Fahrzeug erzeugt und anschließend folgende Vorgänge durchführt:

1.) mit dem Fahrrad losfahren

2.)mit dem Auto losfahren

3.) mit dem Fahrrad anhalten

4.) mit dem Auto anhalten

Unmittelbar nach jedem der vier Vorgänge soll jeweils mittels der Methode status() der aktuelle Fahrzustand beider Fahrzeuge ausgegeben werden. Eine Ausgabe des Testprogramms sollte also etwa so aussehen:

Zustand1:

Fahrrad1 faehrt auf 2 Reifen mit je 4.5 bar

Auto1 steht auf 4 Reifen mit je 1.9 bar

Zustand 2:

Fahrrad1 faehrt auf 2 Reifen mit je 4.5 bar

Auto 1 fährt auf 4 Reifen mit je 1.9 bar

Zustand 3:

Fahrrad1 steht auf 2 Reifen mit je 4.5 bar

Auto1 faehrt auf 4 Reifen mit je 1.9 bar

Zustand 4:

Fahrrad1 steht auf 2 Reifen mit je 4.5 bar

Zusatzübung

Implementiere eine Klasse Radio mit folgenden (*nicht statischen*) Methoden:

- void lauter()/void leiser(): Verändern die Objektvariable lautstärke.
- void an()/void aus()/boolean istAn(): Nutzen die Objektvariable eingeschaltet. Die Methoden an()/aus() sollen Meldungen wie "an"/"aus" auf dem Bildschirm ausgeben. (Optional: Die Lautstärke soll nur im Bereich von 0 bis 100 liegen.)
- void wähleSender(double frequenz): Sie soll eine Frequenz intern in einer neuen Objektvariablen speichern, ähnlich wie die Lautstärke.
- public String toString(): Sie soll Informationen über den internen Zustand als String zurückgeben, wobei die Zeichenkette die Form "Radio an: Freq=234.0, Laut=2" haben sollte. In einer Zusatzklasse instatiieren Sie ein Radio, stellen es an und machen es lauter.

Geben Sie den Zustand des Radioobjektes aus.