static versus non static

Grundprinzip

Felder : enthalten die Werte, welche den Zustand einer Klasse oder Objektes speichern → **Classmember**

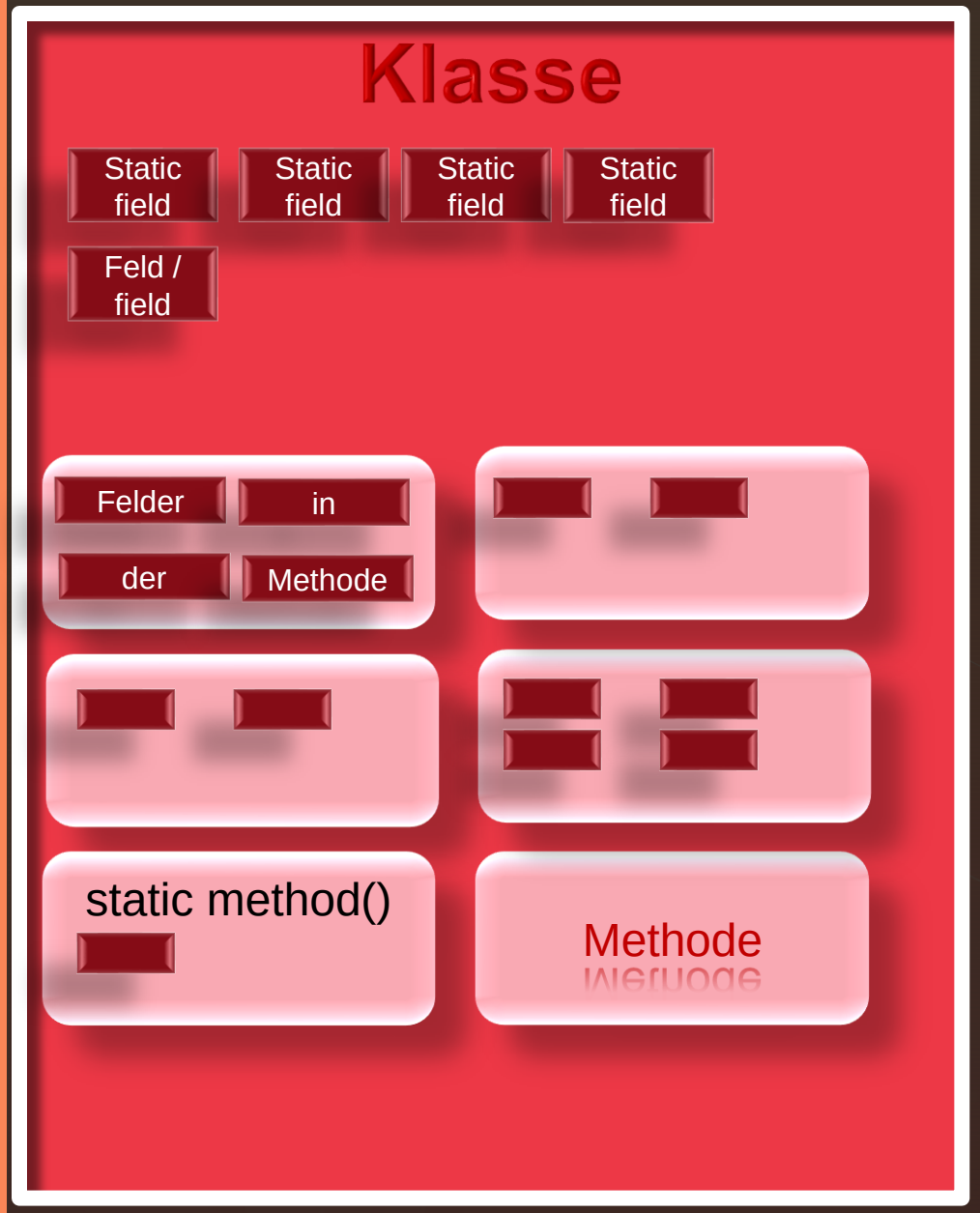
Methoden: bestimmen das Verhalten eines Objektes oder eines Klasse → **Classmember**

Klasse : besteht aus Feldern und Methoden.
→ Classmembern

static field

+

static method



Static field

- Ist an die Klasse gebunden
- Durch das keyword static erkennbar

- Statische Variablen (Felder) sind für die ganze Klasse gültig
- Können von überall verändert werden (überschrieben)

static method()

- Ist an die Klasse gebunden
- Durch das keyword static erkennbar

- Statische Methoden sind innerhalb der gesamten Klasse, aus ebenfalls statischen Methoden, aufrufbar
- Sie bestimmen das Verhalten der Klasse.
- Statische Methoden können statische Felder verwenden und verändern, so daß diese auch in anderen statischen Methoden mit den veränderten Werten erreichbar sind

static → Bedeutet eine Bindung an die Klasse !!

```
class Beispiel{
    static int zahl2 = 5;
    static String text = “Ergebnis“ ;

    static void main(String[] args){
    }
}
```

Festlegen der an die Klasse gebundenen Variablen, innerhalb der Klasse Beispiel

Durch das keyword **static** teile ich Java mit, das diese Variablen für die ganze Klasse nutzbar und erreichbar sind.

Sie können über :

- *Beispiel.zahl2 ; → innerhalb der Klasse und des Packets aufgerufen werden (da kein Zugriffsmodifizierer angegeben wurde → package private)*
- *Innerhalb einer statischen Methode der Klasse über den Bezeichner zahl2;*

Zur Nutzung der Klasse bedarf es der **IMMER statischen** main() Methode

```
class Beispiel{
    static int zahl2 = 5;
    static String text = “Ergebnis“ ;

    static void addieren(){


    }

    static void main(String[] args){

    }
}
```

OHNE Zugriffsmodifizierer (access modifier) ist jede Methode oder Feld *innerhalb der Klasse oder dem Paket* in welchem die Klasse ist *sichtbar*.

Erstellen einer Methode welche einfach zwei Werte addieren soll . Diese Methode ist durch static über die Klasse erreichbar

- Da die Methode addieren() durch **static** an die Klasse gebunden ist, kann sie jetzt von *JEDER statischen Methode innerhalb und ausserhalb der Klasse* aufgerufen werden.
- Von *ausserhalb MUSS* der Klassenname dazugeschrieben werden : **Beipiel.addieren()**
- Innerhalb der Klasse kann , aber muss ich den Klassennamen nicht vorneweg schreiben

```
class Beispiel{
    static int zahl2 = 5;
    static String text = "Ergebnis" ;

    static void addieren(){
        int wert = 2;

        zahl2 = wert + zahl2
    }

    static void main(String[] args){
        addieren();

    }
}
```

Das Keyword **void** bedeutet, das diese Methode nur ausgeführt wird, und nach Ihrer Ausführung wieder an die aufrufende Methode zurückspringt

Erstellen einer lokalen Variable wert mit dem Wert 2

Die Klassenvariable zahl2 kann hier benutzt werden, da sowohl zahl2 als auch die Methode statisch ist. Sie wird mit der lokalen Variable addiert

- Innerhalb der addieren() methode wird die **Klassenvariable** (**static**) zahl2 verändert .
  → Diese Änderung ist für **jeden jetzt gültig**

- **Der Wert in zahl2 ist hiernach 7. Die Initialisierung mit dem Wert 5 ist jetzt überschrieben**

Aufruf der neu erstellten addieren() Methode über die main()

```
class Beispiel{
    static int zahl2 = 5;
    static String text = "Ergebnis" ;

    static void addieren(){
        int wert = 2;
        zahl2 = wert + zahl2
    }

    static void ausgeben(){


    }

    static void main(String[] args){
        addieren();

    }
}
```

Erstellen einer weiteren statischen methode zur Ausgabe des Wertes von zahl2 und der text Variablen

```
class Beispiel{
    static int zahl2 = 5;
    static String text = "Ergebnis" ;

    static void addieren(){
        int wert = 2;
        zahl2 = wert + zahl2
    }

    static void ausgeben(){

        System.out.println( text + zahl2 );

    }

    static void main(String[] args){
        addieren();
        ausgeben();
    }
}
```

Die Methode ausgeben() wird nach der addieren() Methode aufgerufen. Das bedeutet, daß zuerst die Werte in der addieren() methode geändert werden, und im Anschluss die geänderten Werte auf der Konsole ausgegeben werden.

Ausgabe:

```
Ergebnis 7
```

In der Methode ausgeben() soll nur auf die Klassenvariablen zugegriffen werden und deren Wert auf der Konsole ausgegeben werden.

In der Methode main() wird jetzt die statische Methode ausgeben() aufgerufen.

```
class Beispiel{
    static int zahl2 = 5;
    static String text = "Ergebnis" ;

    static void addieren(){
        Int wert = 2;
        zahl2 = wert + zahl2
    }

    static void ausgeben(){

        System.out.println( text + zahl2 );

    }

    static void main(string[] args){
        addieren();
        ausgeben();
    }
}
```

Fazit : static

- ✓ Alles was mit static definiert wird im Sourcecode, ist IMMER an die Klasse gebunden und kann nur von statischen Klassenmembern *(Klassenmember = Felder und Methoden)* genutzt und gesehen werden
- ✓ Innerhalb einer statischen Methode können **NUR statische Felder, lokale Felder** verwendet und geändert werden
- ✓ Eine statische Methode kann **NUR statische Methoden** sehen und aufrufen
- ✓ Von ausserhalb der Klasse kann auf die statischen Klassenmember **NUR mittels des Klassennamen gefolgt von einem Punkt zugegriffen werden** (abhängig vom Zugriffsmodifizierer).
- ✓ → Beispiel.zahl2; Beispiel.text; Beispiel.addieren(); oder Beispiel.ausgeben();

non static field

+

non static method

Instanz Feld

- Ist an das Objekt (Instanz) gebunden
- das keyword static fehlt hier

- Instanz Variablen (Felder) sind nur für den Status der Objekte gültig
- Die Werte einer Instanzvariablen beschreiben die Eigenschaften des Objektes

method()

- Ist nur für das Objekt sichtbar
- Das keyword static ist hier NICHT vorhanden

- Instanz Methoden können von jedem Objekt der Klasse genutzt werden.
- Sie bestimmen das verhalten der Objekte.
- Instanz Methoden können die Felder der Objekte verwenden und verändern.
- Jedes Objekt hat eigene Werte und nur diese werden in den Instanz Methoden verändert

non static → Bedeutet eine Bindung an das Objekt!!

```
class Kunde{
    String name;
    int alter = 18;
    char gender;

    Kunde(){

    }

}
```

Instanzvariablen, welche die Eigenschaft des Objektes bestimmen

Der Konstruktor erzeugt ein neues Objekt der Klasse

Kunde:

name: null
alter : 18
gender:\u0000

- Jede Klasse hat einen Konstruktor
- Durch das Keyword **new** und dem Aufruf Kunde() wird jedesmal ein neues Objekt der Klasse erstellt
- Gibt es keine Anweisungen innerhalb des Konstruktor, erhält jedes Objekt die in den Instanzvariablen vordefinierten Werte
- Ohne eine Referenzvariable kann das Objekt nur kurzfristig genutzt werden und danach NICHT mehr erreicht werden

```
class Kunde{
    private String name;
    private int alter = 18;
    private char gender;

    Kunde(){
        this.name = "Gast" ;
        this.gender = 'm';
    }
}
```

Kapselung (Encapsulation) der Instanzvariablen mit **private**

Der Konstruktor der Klasse Kunde setzt hier Standardwerte für jedes Objekt der Klasse Kunde fest.

- Ein **wichtiges Prinzip** der Objektorientierten Programmierung (OOP) ist die **Kapselung (Encapsulation)** oder auch Geheimnisprinzip .
- Das wird erreicht, wenn die **Instanzvariablen** durch den Zugriffsmodifizierer **private** nur für Methoden der jeweiligen Klasse sichtbar gemacht werden.
- **Private** bedeutet **KEIN Zugriff von ausserhalb der Klasse** (auch nicht über das Objekt) **ist möglich**.

- Hier werden **innerhalb des Konstruktors JEDEM** neu erstellten **Objekt** der Klasse Kunde Standardwerte zugewiesen.
- Diese Werte sind jetzt für niemanden ausserhalb der Klasse veränderbar!!!

Setter und Getter Methoden

Durch die Kapselung der Instanzvariablen und ihre Erreichbarkeit ausschließlich in derselben Klasse kann bestimmt werden ob und wie Zugriff auf diese gewährt wird.

```
class Kunde{
    private String name;
    private int alter = 18;
    private char gender;

    Kunde(){
        this.name = "John" ;
        this.gender = 'm';
    }

    public void setName(String name) {
        this.name = name
    }

    public String getName() {
        return name ;
    }
}
```

- Die **Setter** Methoden können über das Objekt erreicht werden.
- Sie beziehen sich immer auf **eine einzelne Variable**
- Innerhalb der () wird ihnen der Wert übergeben.
- Er **MUSS** immer vom **selben Typ** der Instanzvariable sein.

Setter Methode über die der Wert der Instanzvariable name individuell für jedes Objekt geändert werden kann

Getter Methode über die der Wert der Instanzvariable name ausgelesen werden kann

- Die **Getter** Methoden dienen ausschließlich dem **Zweck, den Wert einer Instanzvariablen abzurufen**
- Sie beziehen sich auch immer **NUR auf EINE** Variable und müssen **vom selben Typ der Instanzvariable** sein.
- Durch das keyword **return** wird der Wert Methodenübergreifen weitergegeben

```
class Shop{
    static Kunde erstellekunde() {
        Kunde kunde = new Kunde() ;
        return kunde ;
    }
}
```

Aufruf Klasse Kunde

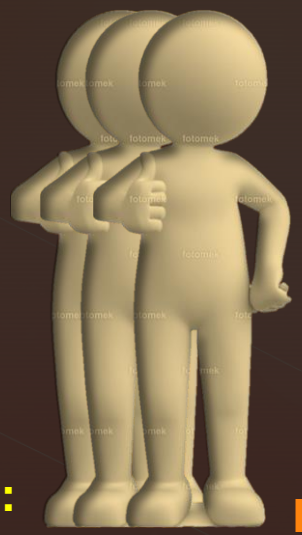
```
class Kunde{
    private String name;
    private int alter = 18;
    private char gender;

    Kunde(){
        this.name = "John" ;
        this.gender = 'm';
    }
}
```

Suche nach passen-dem Konstruktor

Rücksprung zu der aufrufenden Methode

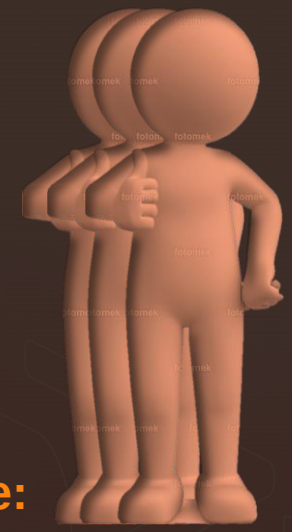
- Es KANN in einer statischen Methode ein Objekt erzeugt werden.
- Dazu muss jedesmal mit new KonstruktorderKlasse() aufgerufen werden.
- Eine Referenzvariable vom Typ der Klasse (oder einer übergeordneten Klasse) hilft das Objekt auf dem Heap zu erreichen.
- Die Objekte können jetzt ALLE non static Instanzvariablen und Methoden nutzen.
- Objekte KÖNNEN auch static Variablen oder Methoden nutzen.

Erstellen des Objektes



Kunde:

name: Peter
alter : 18
gender:m



Kunde:

name: Lena
alter : 35
gender:w

Fazit : non static

- ✓ Alles was ohne static definiert wird im Sourcecode, ist IMMER an die Objekte gebunden und kann nur von Objekten genutzt und gesehen werden
- ✓ Innerhalb einer nicht statischen Methode können **NUR nicht statische Felder, lokale Felder** verwendet und geändert werden
- ✓ Objekte haben **Zugriff auf statische und nicht statische Klassenmember.**
- ✓ Wenn ein **statischer Wert durch ein Objekt geändert** wird, ist er für **ALLE Objekte** geändert, ein ***nicht statischer Wert*** wird immer ***NUR für das referenzierte Objekt verändert***