# 04-Objekte-1-Einführung

Objektorientierte Programmierung | Matthias Tichy

Software Engineering
Programming Languages

universität ulm

# Lernziele

- Objekte
- Klassen
- Methoden
- Die Klasse String
- Überladen
- Statische Attribute und Methoden

# Objekte und Klassen

# Zusammenhang Objekte und Klassen

```
public class String
{
  private byte[] value;
  private byte coder;
  static final byte UTF16  = 1;
  //[...]
  public String(String original) {/*[...]*/}
  public String(char value[]) {/*[...]*/}
  public int length() {/*[...]*/}
  public String toUpperCase() {/*[...]*/}
}
```

vereinfacht

a:String
value:[0,85,0,76,0,77]
coder:1

b:String
value:[0,84,0,105,0,99,0,104,0,121]
coder:1

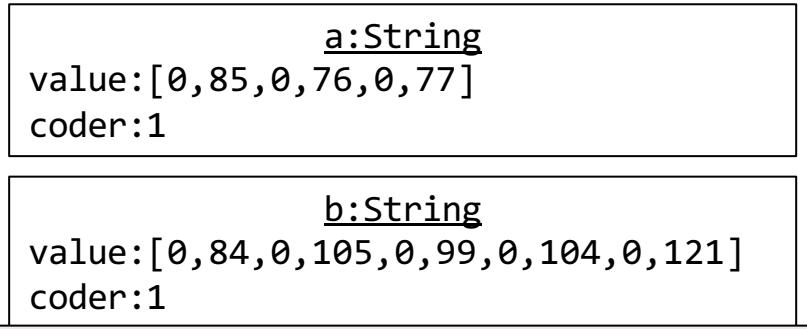
https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/lang/String.java

# Instanziierung

```
public class String
{
  private byte[] value;
  private byte coder;
  static final byte UTF16  = 1;
  //[...]
  public String(String original) {/*[...
  public String(char value[]) {/*[...]*/
  public int length() {/*[...]*/}
  public String toUpperCase() {/*[...]*/
}
```

https://github.com/openjdk/jdk/blob/master/src/java.base/share/cla

vereinfacht




```
jshell> String a = new String ("ULM");
a ==> "ULM"
jshell> a.length()
$9 ==> 3
jshell> String b = new String ("Tichy");
b ==> "Tichy"
jshell> b.length()
$11 ==> 5
```

# Klassen

Integration von Daten und Funktionen

Daten für die Abbildung einer Zeichenkette

Konstruktoren zur Erzeugung von Instanzen

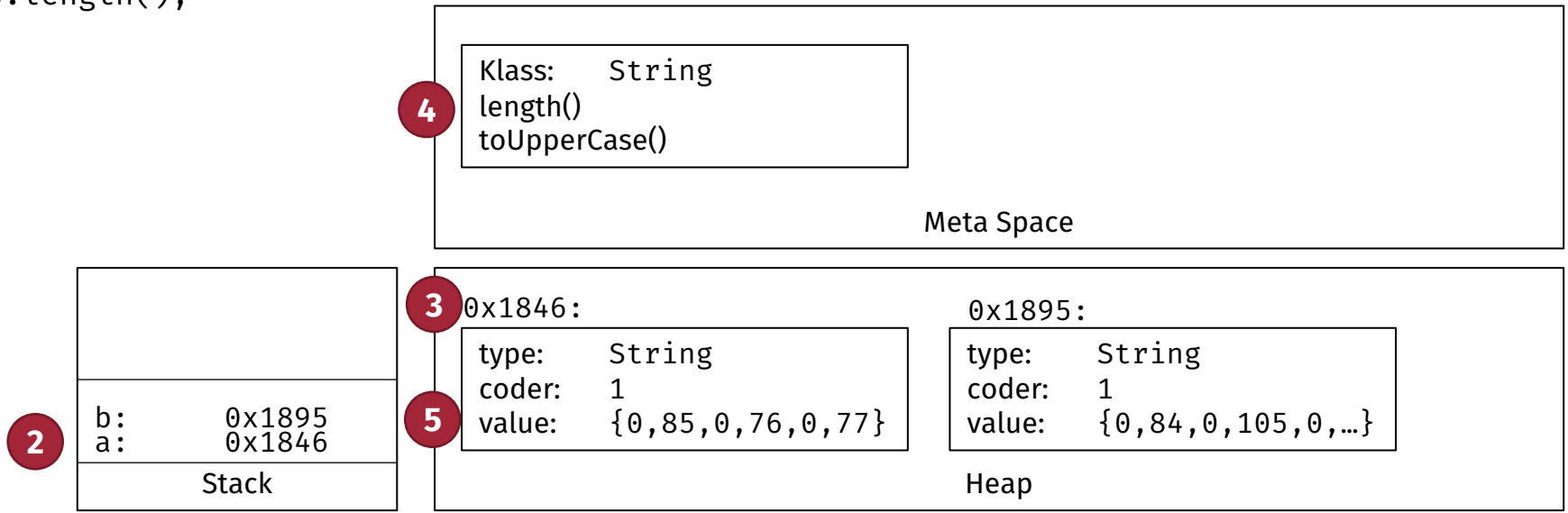
Auf den Daten arbeitenden Methoden

```
public class String
{
  private byte[] value;
  private byte coder;
  static final byte UTF16  = 1;
  //[...]
  public String(String original) {/*[...]*/}
  public String(char value[]) {/*[...]*/}
  public int length() {/*[...]*/}
  public String toUpperCase() {/*[...]*/}
}
```

# Objekte

Methodenaufruf auf Objekt (Daten des Objekts + Methode aus der Klasse)

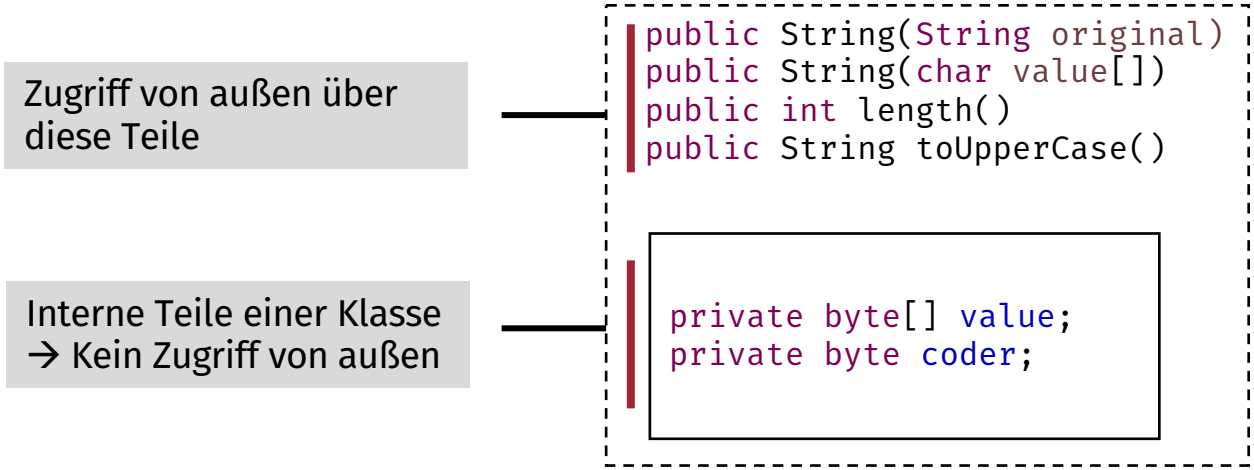
```
String a = new String ("ULM");
a.length();
String b = new String ("Tichy");
b.length();
```



# Kapselung

```
public class String
{
  private byte[] value;
  private byte coder;
  static final byte UTF16  = 1;
  //[...]
  public String(String original) {/*[...]*/}
  public String(char value[]) {/*[...]*/}
  public int length() {/*[...]*/}
  public String toUpperCase() {/*[...]*/}
}
```

# Kapselung

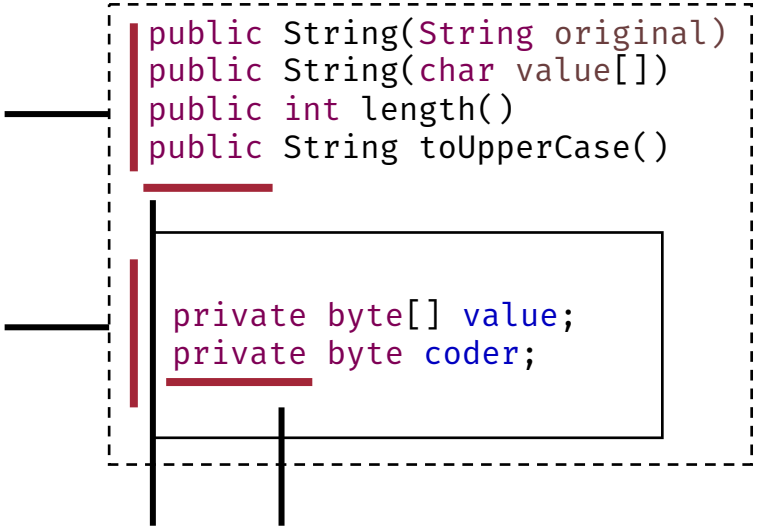



- Trennung Schnittstelle und Implementierung
- Kein Zugriff auf interne Daten und Implementierungsdetails
- Änderung möglich ohne Auswirkungen auf Nutzer

# Kapselung

Zugriff von außen über diese Teile

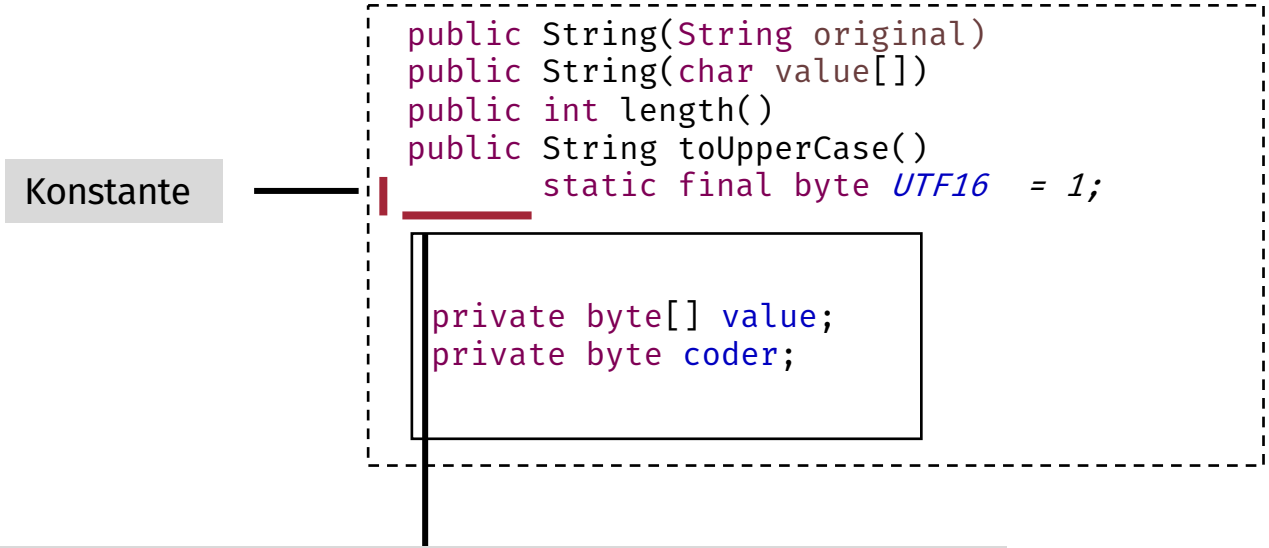
Interne Teile einer Klasse → Kein Zugriff von außen




`public` → von jedem nutzbar (sichtbar)

`private` → nur innerhalb der Klasse nutzbar (sichtbar)

# Kapselung


public String(String original)
public String(char value[])
public int length()
public String toUpperCase()
static final byte UTF16 = 1;
private byte[] value;
private byte coder;
Konstante
<keine Angabe>
→ von jedem innerhalb des gleichen Pakets nutzbar (sichtbar)
Package-Sichtbarkeit

# String

```
String cde = "cde";
System.out.println("abc" + cde);
String s1 = "abc".substring(2, 3);
String s2 = cde.substring(1, 2);
```

- Instanzen der Klasse String sind Konstanten → 5 Strings im obigen Beispiel
- Änderbare Strings →StringBuilder, StringBuffer
- Methoden der Klasse dokumentiert in JavaDoc

https://docs.oracle.com/en/java/javase/20/docs/api/java.base/java/lang/String.html

# Murmelgruppe

Java Playground auf
https://www.online-java.com/TvxdzYsH9q

Nutzen der Methoden der Klasse String

> Schreiben Sie eine statische Methode, die prüft, ob der übergebene String mit Leerzeichen beginnt oder endet.

- Rufen Sie diese Methode in der main-Methode mit einem Beispiel-String auf

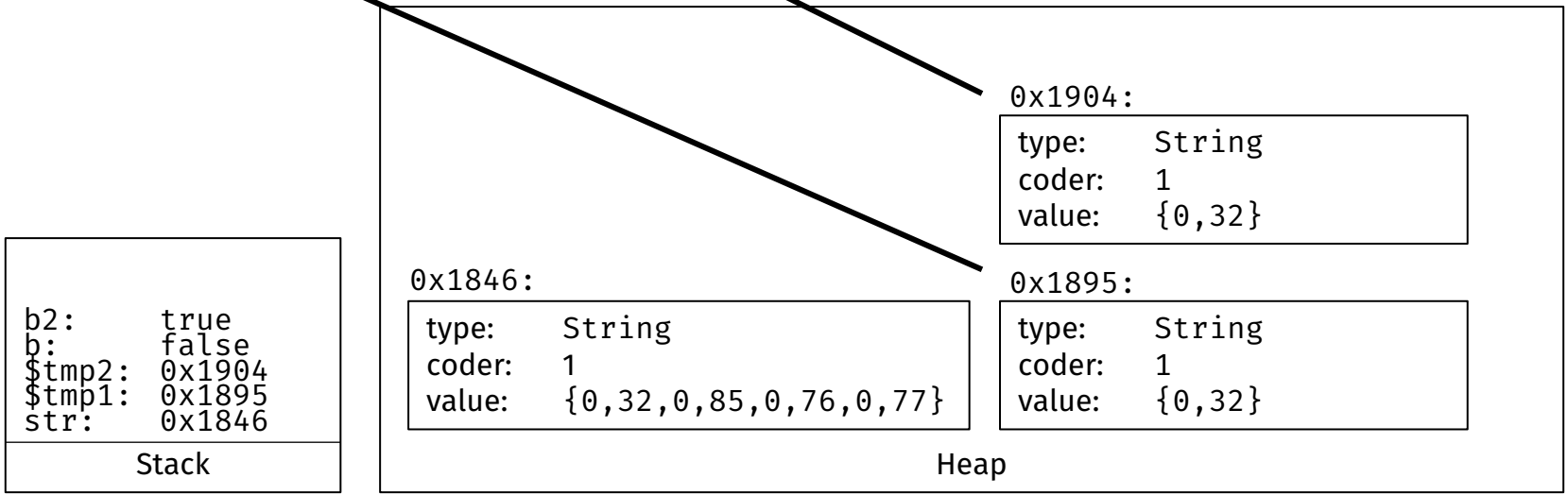
```
public static boolean startsOrEndsWithSpace(final String str) {

 // TODO

}
```

# Gleichheit

```
String str = new String (" ULM");
boolean b = str.substring(0,1) == " ";
boolean b2 = str.substring(0,1).equals(" ");
```



# Gleichheit

== und !=

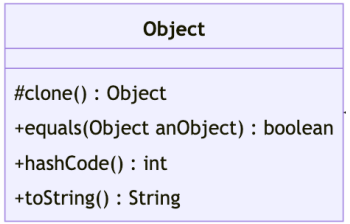
- Referenztypen:
  - Obacht: Vergleich der Referenzen!
  - → „dasselbe Objekt“ vs. „das gleiche Objekt“
  - → Vergleich auf „das gleiche“ Objekte immer mit equals()!

- Primitive Datentypen:
  - Direkter Vergleich der Werte
  - Obacht:
    - Vergleich bei Fließkommazahlen ist schwierig
    - Fließkommazahlen sind allgemein schwierig

Bruce Dawson. **Comparing Floating Point Numbers, 2012 Edition**

# Die Klasse Object

- Alle Java Klassen erben (automatisch) von Object

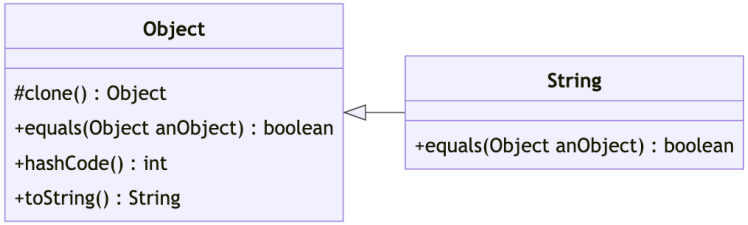



```java
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Methoden können in der erbenden Klasse überschrieben werden (später mehr)

# Die Klasse Object

- Alle Java Klassen erben (automatisch) von Object




```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Methoden können in der erbenden Klasse überschrieben werden (später mehr) → String überschreibt equals()

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    return (anObject instanceof String aString)
            && (!COMPACT_STRINGS || this.coder == aString.coder)
            && StringLatin1.equals(value, aString.value);
}
```

# Überladen

Mehrere Methoden mit gleichem Namen und unterschiedlicher Signatur

```
public String(String original)
public String(char value[])
public int length()
public String toUpperCase()
public String[] split(String regex)
public String[] split(String regex, int limit)
public boolean startsWith(String prefix)
public boolean startsWith(String prefix, int toffset)
```

- Typen der Argumente → Auswahl der aufzurufenden Methode
- Ziel: weniger Methodennamen
- Überschreiben: Ähnliches Konzept später bei Vererbung

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman:
The Java® Language Specification - Java SE 20 Edition – 2023-03-03 - §8.4.9 Overloading
https://docs.oracle.com/javase/specs/jls/se20/html/jls-8.html#jls-8.4.9

# Statische Attribute und Methoden einer Klasse

- Statische Attribute und Methoden einer Klasse
  - Attribute sind für alle Objekte einer Klasse identisch
  - Attribute und Methoden können ohne Objekt direkt auf der Klasse aufgerufen werden

```
public String(String original)
public String(char value[])
public int length()
public String toUpperCase()
static final byte UTF16  = 1;
static String format(String format, Object... args)
```

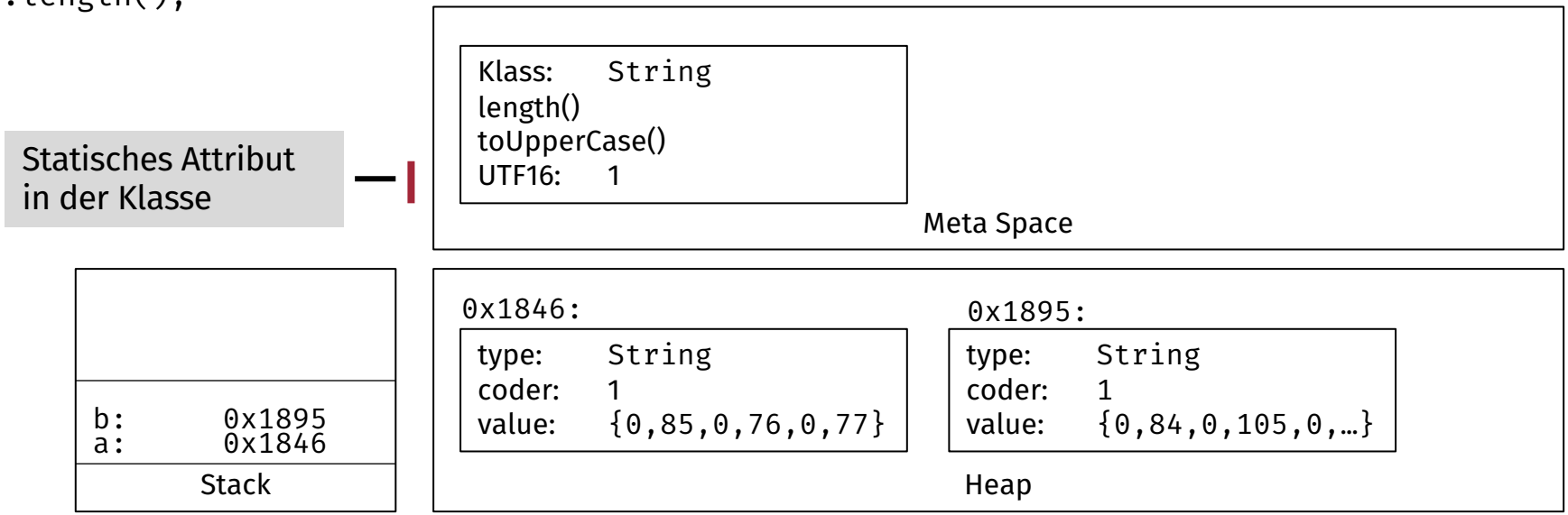
```
private byte[] value;
private byte coder;
```

static
→ unabhängig von Objekten der Klassen

```
String.format("Counter value: %d\n", i);
```

# Statische Attribute

```
String a = new String ("ULM");
a.length();
String b = new String ("Tichy");
b.length();
```



# Lernziele

- Objekte
- Klassen
- Methoden
- Die Klasse String
- Überladen
- Statische Attribute und Methoden