

Lektion 10

Tell, don't ask

non-static- vs. static-Methoden

null-Referenz

this-Referenz II

Automatische Initialisierung

Garbage Collector

Schauen wir uns ein weiteres
Beispiel an!

```
public class Rechteck {
    double laenge;
    double breite;

    public Rechteck(double laenge, double breite) {
        if (laenge > 0 && breite > 0) {
            this.laenge = laenge;
            this.breite = breite;
        }
    }

    public double getLaenge() {
        return laenge;
    }

    public double getBreite() {
        return breite;
    }
}
```

```
public class RechteckTest
{
    public static void main(String[] args)
    {
        Rechteck r = new Rechteck(10.0, 20.0);
        System.out.println(r.breite * r.laenge);
    }
}
```

Was können wir verbessern?

```

public class Rechteck {
    double laenge;
    double breite;

    public Rechteck(double laenge, double breite) {
        if (laenge > 0 && breite > 0) {
            this.laenge = laenge;
            this.breite = breite;
        }
    }

    public double getLaenge() {
        return laenge;
    }

    public double getBreite() {
        return breite;
    }
}

```

```

public class RechteckTest
{
    public static void main(String[] args)
    {
        Rechteck r = new Rechteck(10.0, 20.0);
        System.out.println(r.breite * r.laenge);
    }
}

```

Wir greifen direkt auf die **Attribute** zu. In OO-Sprachen geschieht dies i.d.R. über die getter- und setter-Methoden.

```
public class Rechteck {
    double laenge;
    double breite;

    public Rechteck(double laenge, double breite) {
        if (laenge > 0 && breite > 0) {
            this.laenge = laenge;
            this.breite = breite;
        }
    }

    public double getLaenge() {
        return laenge;
    }

    public double getBreite() {
        return breite;
    }
}
```

```
public class RechteckTest
{
    public static void main(String[] args)
    {
        Rechteck r = new Rechteck(10.0, 20.0);
        System.out.println(r.getBreite() * r.getLaenge());
    }
}
```

Können wir noch weitere Verbesserungen vornehmen?

```

public class Rechteck {
    double laenge;
    double breite;

    public Rechteck(double laenge, double breite) {
        if (laenge > 0 && breite > 0) {
            this.laenge = laenge;
            this.breite = breite;
        }
    }

    public double getLaenge() {
        return laenge;
    }

    public double getBreite() {
        return breite;
    }
}

```

```

public class RechteckTest
{
    public static void main(String[] args)
    {
        Rechteck r = new Rechteck(10.0, 20.0);
        System.out.println(r.getBreite() * r.getLaenge());
    }
}

```

Oben überlassen wir es dem Programmierer, der unsere Klasse nutzt, den Flächeninhalt zu berechnen.

Warum lassen wir unsere Klasse nicht selbst die Berechnung vornehmen?

```

public class Rechteck {
    double laenge;
    double breite;

    public Rechteck(double laenge, double breite) {
        if (laenge > 0 && breite > 0) {
            this.laenge = laenge;
            this.breite = breite;
        }
    }

    public double berechneFlaeche() {
        return laenge*breite;
    }
}

```

```

public class RechteckTest
{
    public static void main(String[] args)
    {
        Rechteck r = new Rechteck(10.0, 20.0);
        System.out.println(r.berechneFlaeche());
    }
}

```

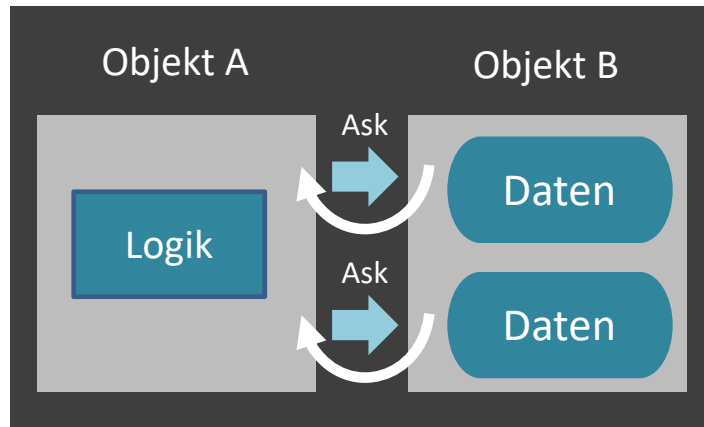
Rechteck kapselt die Berechnung in einer eigenen Methode.

Der Zugriff auf die Attribute über die getter ist nicht mehr notwendig.

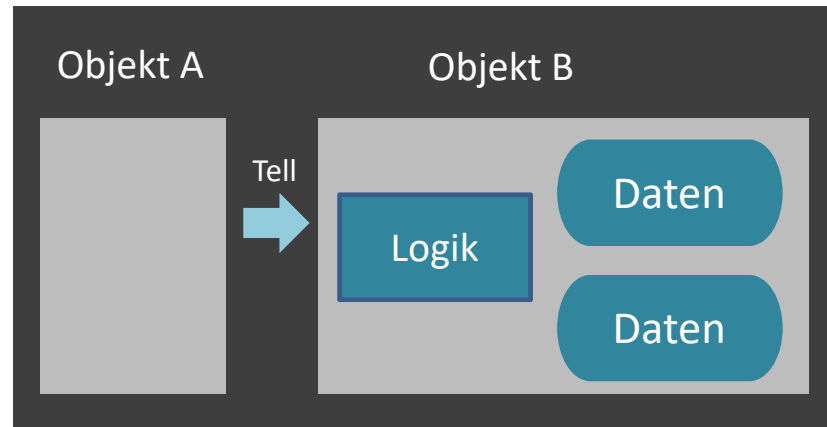
Wir fragen das Objekt nicht mehr nach seinen Daten, um eine Berechnung durchzuführen, sondern lassen es die Berechnung selbst machen.

➤ Dieses Prinzip heißt auch **Tell, don't ask**

Tell, don't ask!



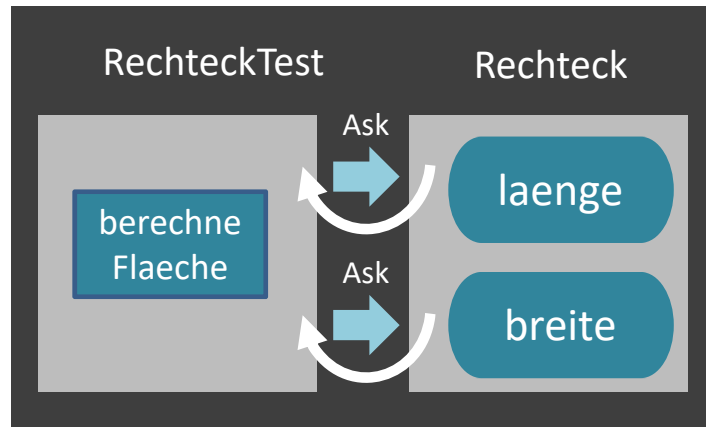
Wenn Objekt A intensiv ein Objekt B nach Daten fragt, um auf Basis der Daten eine Entscheidung/Berechnung durchzuführen, wurde offensichtlich Programmlogik und Datenhaltung getrennt.



Wenn Objekt A Objekt B nur mitteilt, „was“ getan werden soll – und nicht „wie“, dann liegen Programmlogik und Daten zusammen in einem Objekt.

➤ Dies vereinfacht i.d.R. die Entwicklungsarbeit!

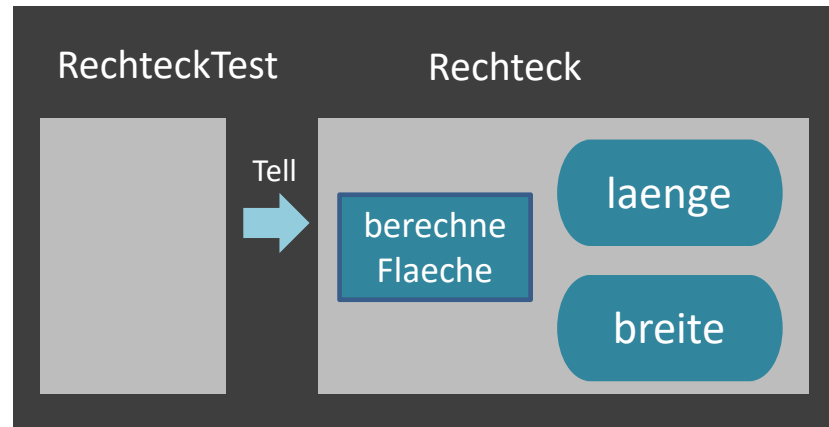
Tell, don't ask! - Beispiel



Was soll getan werden: berechneFlaeche

Wie soll die Fläche berechnet werden:

- laenge und breite erfragen
- laenge und breite multiplizieren



Was soll getan werden: berechneFlaeche

Wie soll die Fläche berechnet werden?

- Das ist die Sache des Rechtecks.

- Die Verwendung der Rechteck-Klasse ist aus Sicht eines Entwicklers einfacher!

Klassenvariablen und –methoden static

Modifier static

- Durch den Modifier `static` wird ein Attribut oder eine Methode **der Klasse und nicht dem Objekt zugeordnet**.
- Wird benutzt, wenn eine Methode unabhängig vom Zustand des Objekts ist.
- Beispiele:
 - Die statische Methode `Math.sin(double x)` verarbeitet eine `double` Variable zu deren Sinuswert, unabhängig von einem „(Zwischen)“-Zustand. Es wird kein Objekt benötigt.
 - Die statische Variable `Math.PI` ist eine konstante Zahl, die unabhängig vom Zustand eines Objekts „lebt“.
 - Ein Objekt jedes neuen Produkts erhält eine Seriennummer, wenn es erstellt wird. Die Klasse verfügt über die nächste zu vergebende Seriennummer, die bei der Erstellung jedes Produkts zugewiesen und hochgezählt wird.
 - `static` wird für Singletons benutzt (später).

```

public class Pizza {
    static int anzahlGebackenePizzen = 0;

    String name;
    int durchmesser;
    float preis;

    public Pizza(String name, int durchmesser, float preis) {
        this.name = name;
        this.durchmesser = durchmesser;
        this.preis = preis;
        anzahlGebackenePizzen++;
    }

    public static int ermittleLaengstenNamen(Pizza[] pizzas) {
        int laengsterName = 0;
        for (int i = 0; i < pizzas.length; i++) {
            int laengeAktuellerPizza = pizzas[i].getName().length();
            if (laengsterName < laengeAktuellerPizza)
                laengsterName = laengeAktuellerPizza;
        }
        return laengsterName;
    }
    // getter und setter
    ...
}

```

Aufruf von der Speisekarte-Klasse:

```

int laengsterName =
    Pizza.ermittleLaengstenNamen(pizzas);

```

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;  
  
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza(String name, int durchmesser) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        anzahlGebackenePizzen++;  
    }
```

ggf. steht beim Backen der Pizza
der Preis noch nicht fest
(z. B. wegen Firmenrabatten)

```
    public Pizza(String name, int durchmesser, float preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
        anzahlGebackenePizzen++;  
    }  
    // getter und setter  
    ...
```

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;  
  
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza(String name, int durchmesser) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        anzahlGebackenePizzen++;  
    }
```

anzahlGebackenePizzen wird
bei der Erstellung jeder Pizza
erhöht.

```
    public Pizza(String name, int durchmesser, float preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
        anzahlGebackenePizzen++;  
    }  
    // getter und setter  
    ...
```

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza(String name, int durchmesser) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
        anzahlGebackenePizzen++;  
    }  
    // getter und setter  
    ...
```

anzahlGebackenePizzen wird
bei der Erstellung jeder Pizza
erhöht.



doppelter Code



this

- Mit **this** können aus einem Konstruktor heraus auch andere **Konstruktoren** des gleichen Objekts **aufgerufen** werden.

```
public class PersonNr {  
    String name;  
    int personalNummer;
```

```
    public PersonNr(String name) {  
        this(name, -1);  
    }
```

this() muss die erste
Anweisung im Konstruktor sein

```
    public PersonNr(String name, int personalNummer) {  
        this.name = name;  
        this.personalNummer = personalNummer;  
    }  
}
```



```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;  
  
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza(String name, int durchmesser) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        anzahlGebackenePizzen++;  
    }
```

anzahlGebackenePizzen wird
bei der Erstellung jeder Pizza
erhöht.

```
    public Pizza(String name, int durchmesser, float preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
        anzahlGebackenePizzen++;  
    }  
    // getter und setter  
    ...
```

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
    // getter und setter  
    ...
```

Daher kann
anzahlGebackenePizzen
in den Standardkonstruktor
ausgelagert werden.

Standardkonstruktor kann
durch Verwendung von
this() aufgerufen werden.

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
    // getter und setter  
    ...
```

Der hervorgehobene Code ist
doppelt.



Lässt sich vermeiden durch
Setzung eines
Standardparameters für den preis.

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this(name, durchmesser, 0);  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
    // getter und setter  
    ...
```

Der hervorgehobene Code ist
doppelt.



Lässt sich vermeiden durch
Setzung eines Standardwertes für
den **preis**.

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this(name, durchmesser, 0);  
  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
    // getter und setter  
    ...
```

Beim Aufruf von
new Pizza("Pizza Salami", 26);
passiert:

```
this("Pizza Salami", 26, 0);  
this();  
anzahlGebackenePizzen++;  
this.name = "Pizza Salami";  
this.durchmesser = 26;  
this.preis = 0;
```

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    public Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this(name, durchmesser, 0);  
  
    }
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }
```

```
    // getter und setter
```

```
    ...
```

Wie kann vermieden werden, dass
eine Pizza ohne Angabe von Namen
und Durchmesser
„gebacken“/erstellt wird?

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;
```

```
    String name;  
    int durchmesser;  
    float preis;
```

```
    private Pizza() {  
        anzahlGebackenePizzen++;  
    }
```

```
    public Pizza(String name, int durchmesser) {  
        this(name, durchmesser, 0);
```

```
}
```

```
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }
```

```
    // getter und setter
```

```
    ...
```

Wie kann vermieden werden, dass
eine Pizza ohne Angabe von Namen
und Durchmesser
„gebacken“/erstellt wird?



Standardkonstruktor **private** machen

Instanzattribut vs. Klassenattribut

```
public class Pizza {  
    static int anzahlGebackenePizzen = 0;    Klassenattribut/Klassenvariable  
  
    String name;  
    int durchmesser;  
    float preis;  
  
    public Pizza() {                          Standardkonstruktor/Default constructor  
        anzahlGebackenePizzen++;  
    }  
  
    public Pizza(String name, int durchmesser, float preis) {  
        this();  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }  
    // getter und setter  
    ...  
}
```


Sichtbarkeit von Variablen

```
public class Rechteck {  
    private double laenge;  
    public double breite;  
    public static int anzahlRechtecke = 0;
```

```
    public Rechteck(double laenge, double breite) {  
        if (laenge > 0 && breite > 0) {  
            this.laenge = laenge;  
            this.breite = breite;  
        }  
        anzahlRechtecke++;  
    }
```

```
    public double berechneFlaeche() {  
        double area;  
        area = laenge*breite;  
        return area;  
    }  
}
```

```
public class TestRechteck  
{  
    public static void main(String[] args)  
    {  
        Rechteck r;  
        r = new Rechteck(10.0, 20.0);  
        System.out.println(r.breite);  
        System.out.println(Rechteck.anzahlRechtecke);  
    }  
}
```

Sichtbarkeit von Variablen

```
public class Rechteck {  
    private double laenge; ← Instanzattribut  
    public double breite;  
    public static int anzahlRechtecke = 0; ← Klassenattribut  
  
    public Rechteck(double laenge, double breite) ← Methodenparameter {  
        if (laenge > 0 && breite > 0) {  
            this.laenge = laenge;  
            this.breite = breite;  
        }  
        anzahlRechtecke++;  
    }  
  
    public double berechneFlaeche() {  
        double area; ← lokale Variable  
        area = laenge*breite;  
        return area;  
    }  
}
```

```
public class TestRechteck  
{  
    public static void main(String[] args)  
    {  
        Rechteck r;  
        r = new Rechteck(10.0, 20.0);  
        System.out.println(r.breite);  
        System.out.println(Rechteck.anzahlRechtecke);  
    }  
}
```

Sichtbarkeit von Variablen

Lokale Variable area

```
public class Rechteck {  
    private double laenge;  
    public double breite;  
    public static int anzahlRechtecke = 0;  
  
    public Rechteck(double laenge, double breite) {  
        if (laenge > 0 && breite > 0) {  
            this.laenge = laenge;  
            this.breite = breite;  
        }  
        anzahlRechtecke++;  
    }  
  
    public double berechneFlaeche() {  
        double area;          area  
        area = laenge*breite;  
        return area;  
    }  
}
```

Sichtbarkeit von Variablen

Methodenparameter laenge, breite

```
public class Rechteck {  
    private double laenge;  
    public double breite;  
    public static int anzahlRechtecke = 0;  
  
    public Rechteck(double laenge, double breite) {  
        if (laenge > 0 && breite > 0) {  
            this.laenge = laenge;  
            this.breite = breite;  
        }  
        anzahlRechtecke++;  
    }  
  
    public double berechneFlaeche() {  
        double area;  
        area = laenge*breite;  
        return area;  
    }  
}
```

Methodenparameter
laenge und breite

Sichtbarkeit von Variablen

(privates) Instanzattribut laenge

```
public class Rechteck {
```

```
    private double laenge;           Instanzattribut  
    public double breite;           laenge
```

```
    public static int anzahlRechtecke = 0;
```

```
    public Rechteck(double laenge, double breite) {
```

```
        if (laenge > 0 && breite > 0) {           Instanzattribut  
            this.laenge = laenge;                 laenge  
            this.breite = breite;  
        }  
        anzahlRechtecke++;
```

```
    }
```

```
    public double berechneFlaeche() {
```

```
        double area;                               Instanzattribut  
        area = laenge*breite;                       laenge  
        return area;
```

```
    }
```

```
}
```

Sichtbarkeit von Variablen

(öffentliches) Instanzattribut breite

```
public class Rechteck {
```

```
    private double laenge;
```

Instanzattribut

```
    public double breite;
```

breite

```
    public static int anzahlRechtecke = 0;
```

```
    public Rechteck(double laenge, double breite) {
```

```
        if (laenge > 0 && breite > 0) {
```

Instanzattribut

```
            this.laenge = laenge;
```

breite

```
            this.breite = breite;
```

```
        }
```

```
        anzahlRechtecke++;
```

```
    }
```

```
    public double berechneFlaeche() {
```

```
        double area;
```

Instanzattribut

```
        area = laenge*breite;
```

breite

```
        return area;
```

```
    }
```

```
}
```

```
public class TestRechteck
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        Rechteck r;
```

Instanzattribut

```
        r = new Rechteck(10.0, 20.0);
```

breite

```
        System.out.println(r.breite);
```

```
        System.out.println(Rechteck.anzahlRechtecke);
```

```
    }
```

```
}
```

Sichtbarkeit von Variablen

(öffentliches) Klassenattribut `anzahlRechtecke`

```
public class Rechteck {
```

```
    private double laenge;
```

```
    public double breite;
```

```
    public static int anzahlRechtecke = 0;
```

Klassenattribut

anzahlRechtecke

```
    public Rechteck(double laenge, double breite) {
```

```
        if (laenge > 0 && breite > 0) {
```

```
            this.laenge = laenge;
```

```
            this.breite = breite;
```

```
        }
```

```
        anzahlRechtecke++;
```

Klassenattribut

anzahlRechtecke

```
    }
```

```
    public double berechneFlaeche() {
```

```
        double area;
```

```
        area = laenge*breite;
```

```
        return area;
```

```
    }
```

```
}
```

Klassenattribut

anzahlRechtecke

```
public class TestRechteck
```

```
{
```

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

```
        Rechteck r;
```

```
        r = new Rechteck(10.0, 20.0);
```

```
        System.out.println(r.breite);
```

```
        System.out.println(Rechteck.anzahlRechtecke);
```

Klassenattribut

anzahlRechtecke

```
    }
```

```
}
```

static – non-static Vergleich

```
public class MyString {
    private char[] internal;

    public MyString(String s) {
        internal = s.toCharArray();
    }

    public static int compare(MyString s1, MyString s2) {
        char[] first = s1.internal;
        char[] other = s2.internal;
        int len = (first.length < other.length)
            ? first.length : other.length;
        for (int i = 0; i < len; i++) {
            if (first[i] != other[i]) {
                return first[i] - other[i];
            }
        }
        return first.length - other.length;
    }
}
```

```
public class MyString {
    private char[] internal;

    public MyString(String s) {
        internal = s.toCharArray();
    }

    public int compareTo(MyString otherMyString) {
        char[] other = otherMyString.internal;

        int len = (internal.length < other.length)
            ? internal.length : other.length;
        for (int i = 0; i < len; i++) {
            if (internal[i] != other[i]) {
                return internal[i] - other[i];
            }
        }
        return internal.length - other.length;
    }
}
```


Es lassen sich auch komplexere Strukturen abbilden, indem man Objekte anderer (selbst definierter) Klassen als Attribute nutzt:

```
public class Adresse {
    String strasse;
    int plz;
    String wohnort;

    public String getStrasse() {
        return strasse;
    }
    public void setStrasse(String strasse) {
        this.strasse = strasse;
    }
    public int getPlz() {
        return plz;
    }
    //weitere getter und setter
    ...
}
```

```
public class Person {
    String vorname;
    String nachname;
    Adresse adresse;

    public Person(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }
    public Adresse getAdresse() {
        return adresse;
    }
    public void setAdresse(Adresse adresse) {
        this.adresse = adresse;
    }
    //getter und setter für vorname und nachname
    ...
}
```

Wenn ich über den **Konstruktor** eine Person anlege, wie kann ich später überprüfen, ob bereits eine Adresse angelegt wurde?

null-Referenz

- null ist eine spezielle Referenz, die genutzt wird, wenn eine Referenz (explizit) auf **kein** Objekt verweist.

- z.B. bei der Initialisierung...

```
public class PersonNr {  
    String name = null;  
    ...  
}
```

- ... oder um eine Referenz auf ein Objekt explizit zu löschen:

```
name = null;
```

- **Beispiel:** Durch `if (name == null)` wird verglichen, ob die Objektreferenz `name` gleich der null-Referenz ist (d.h. auf kein Objekt verweist).

```

public class Adresse {
    String strasse;
    int plz;
    String wohnort;

    public String getStrasse() {
        return strasse;
    }
    public void setStrasse(String strasse) {
        this.strasse = strasse;
    }
    public int getPlz() {
        return plz;
    }
    //weitere getter und setter
    ...
}

```

```

public class Person {
    String vorname;
    String nachname;
    Adresse adresse = null;

    public Person(String vorname, String nachname) {
        this.vorname = vorname;
        this.nachname = nachname;
    }
    public Adresse getAdresse() {
        return adresse;
    }
    public void setAdresse(Adresse adresse) {
        this.adresse = adresse;
    }
    //getter und setter für vorname und nachname
    ...

    public static void main(String[] args) {
        Person p = new Person("Joe", "Cool");
        ...
        if (p.getAdresse() == null) {
            Adresse adresse = new Adresse();
            adresse.setStrasse("1770 James Ave.");
            adresse.setWohnort("St. Paul, Minnesota");
            adresse.setPlz(55105);
            p.setAdresse(adresse);
        }
    }
}

```

null-Referenz Beispiel

```
public String findDownloadLink(String webseite)
{
    ...
    return null; //if no download link is available
}
```

- Es ist auch denkbar statt `null` ...
 - eine Exception zu werfen oder
 - den leeren String `""` zurückzugeben.
- Das Rückgabeverhalten sollte im Java-Doc dokumentiert sein!

```
String s = findDownloadLink("http://geek-and-poke.com/");
if (s != null)
{
    ...
}
```

Automatische Initialisierung


Automatische Initialisierung

Mit einem Standardwert initialisiert werden

- alle **Instanzattribute**, wenn das Objekt erstellt wird.
- alle **Klassenattribute** (static-Attribute), wenn die Klasse geladen wird (i. d. R. bei erster Nutzung).
- alle Elemente eines Arrays bei dessen Erstellung (z. B. `new int[5]`).

Datentyp	Standardwert
boolean	false
char	'\u0000'
byte	(byte) 0
short	(short) 0
int	0
long	0L
float	0.0F
double	0.0
Referenztyp (Objektreferenz, Arrayreferenz, etc.)	null

Person
Pizza[]
Adresse
String
etc.



Automatische Initialisierung von Objektattributen

```
public class Pizza {
```

```
    String name;  
    int durchmesser;  
    float preis;  
    //getter und setter  
    ...
```

```
    public static void main(String[] args) {  
        Pizza p = new Pizza();  
        System.out.println(p.getName());  
        System.out.println(p.getDurchmesser());  
        System.out.println(p.getPreis());  
    }  
}
```

Ausgabe:

null

0

0.0

Automatische Initialisierung von Klassenattributen

```
public class Pizza
{
    static int anzahlGebackenePizzen;
    ...
    public static void main(String[] args)
    {
        System.out.println(Pizza.anzahlGebackenePizzen);
    }
}
```

Ausgabe:

0

Automatische Initialisierung von Arrays

```
public class Pizza
{
    ...
    public static void main(String[] args)
    {
        Pizza[] pizzas = new Pizza[5];
        for (int i = 0; i < pizzas.length; i++)
        {
            System.out.println(pizzas[i]);
        }
    }
}
```

Ausgabe:

null
null
null
null
null

Lokale Variablen werden nicht automatisch initialisiert

```
public static void main(String[] args)
{
    Pizza[] pizzas;
    System.out.println(pizzas);
    pizzas = new Pizza[5];
    for (int i = 0; i < pizzas.length; i++)
    {
        System.out.println(pizzas[i]);
    }
}
```



The local variable **pizzas** may not have been initialized

Garbage Collector

Garbage Collector

- Die Java Virtual Machine verfügt über einen Garbage-Collector.
- Wird ein Objekt von einem laufenden Java-Programm nicht mehr referenziert, gibt der Garbage Collector den dazugehörigen Speicher frei.
- Eine Referenz auf ein Objekt verliert i.d.R. ihre Gültigkeit am Ende des Blocks, in dem sie deklariert wurde.
- Ist die Referenz ein Instanzattribut, verliert sie ihre Gültigkeit, wenn das Objekt selbst nicht mehr gültig ist.