Karlsruher Institut für Technologie

3. Tutorium

Enums , Strings, Konstruktoren, Methoden, Code Stil

Tutorium 14

Péter Bohner | 16.11.2022

Inhaltsverzeichnis

1. Wiederholung
2. Datentypen III
3. Strings
3.1 Vergleiche
4. Konstruktoren
5. Methoden
6. Sauber Programmieren

- Falls ihr SSH für die Abgaben nutzen wollt, müsst ihr bei Artemis einen Passwort-Reset machen.
- https://artemis.praktomat.cs.kit.edu/account/reset/request

- Falls ihr SSH für die Abgaben nutzen wollt, müsst ihr bei Artemis einen Passwort-Reset machen.
- https://artemis.praktomat.cs.kit.edu/account/reset/request

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...
x = x + 1; y = x;

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...
x = x + 1; y = x;
Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

```
null
```


Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...
x = x + 1; y = x;
Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?
null

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

`null`

Was passiert bei folgender Code-Ausführung: `Rectangle rec = null; rec.a = 20f;`**?**

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...
x = x + 1; y = x;
Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?
null
Was passiert bei folgender Code-Ausführung: Rectangle rec = null; rec.a = 20f;?

Wiederholung

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?

Modulo: % Beispiel: 5 % 3 ergibt 2

y = ++x entspricht...

x = x + 1; y = x;

Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?

`null`

Was passiert bei folgender Code-Ausführung: `Rectangle rec = null; rec.a = 20f;`**?**

NullPointerException

Mit welchem Operator berechnet man den Rest der Ganzzahldivision?
Modulo: % Beispiel: 5 % 3 ergibt 2
y = ++x entspricht...
x = x + 1; y = x;
Welches Schlüsselwort steht für eine Referenz auf "kein Objekt"?
null
Was passiert bei folgender Code-Ausführung: Rectangle rec = null; rec.a = 20f;?
NullPointerException

Aufzählungs-Datentyp

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: `enum Name { Liste der Werte }`

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: enum Name { Liste der Werte }

Aufzählungs-Datentyp

Karlsruher Institut für Technologie

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: `enum Name { Liste der Werte }`

Beispiele

```
enum CompassDirection { NORTH, EAST, SOUTH, WEST}
```


Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: enum Name { Liste der Werte }
Beispiele
enum CompassDirection { NORTH, EAST, SOUTH, WEST}

Aufzählungs-Datentyp

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: `enum Name { Liste der Werte }`

Beispiele

```
enum CompassDirection { NORTH, EAST, SOUTH, WEST}
```

Color.java

```
enum Color {
    RED,
    GREEN,
    BLUE
}
```

Enums können in eine eigene Java-Datei oder innerhalb einer Klasse definiert werden.

Aufzählungs-Datentyp

Karlsruher Institut für Technologie

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: `enum Name { Liste der Werte }`

Beispiele

```
enum CompassDirection { NORTH, EAST, SOUTH, WEST}
```

Color.java

```
enum Color {
    RED,
    GREEN,
    BLUE
}
```

Enums können in eine eigene Java-Datei oder innerhalb einer Klasse definiert werden.

Operationen: Müssen vom Benutzer angegeben werden analog zu Klassenmethoden

Eine Liste der möglichen Werte des Typs wird explizit angegeben Schema: enum Name { Liste der Werte }
Beispiele
enum CompassDirection { NORTH, EAST, SOUTH, WEST}
Color.java
enum Color {
RED,
GREEN,
BLUE
}
Enums können in eine eigene Java-Datei oder innerhalb einer Klasse definiert werden.
Operationen: Müssen vom Benutzer angegeben werden analog zu Klassenmethoden

Konstanten

Variablem mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort **`final`** gekennzeichnet:

Variablem mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort final gekennzeichnet:

Konstanten

Karlsruher Institut für Technologie

Variablem mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort **`final`** gekennzeichnet:

lokale Variablen und Attribute

```
final Typ variablenName;
```
oder mit Initialisierung:
```
final Typ variablenName = Wert;
```


Variablem mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort final gekennzeichnet:
lokale Variablen und Attribute
final Typ variablenName;
oder mit Initialisierung:
final Typ variablenName = Wert;

Konstanten

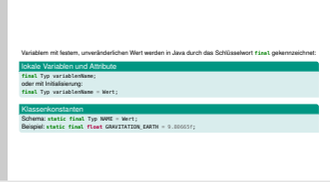
Variablem mit festem, unveränderlichen Wert werden in Java durch das Schlüsselwort **`final`** gekennzeichnet:

lokale Variablen und Attribute

```
final Typ variablenName;
```
oder mit Initialisierung:
```
final Typ variablenName = Wert;
```

Klassenkonstanten

Schema: `static final Typ NAME = Wert;`
Beispiel: `static final float GRAVITATION_EARTH = 9.80665f;`



Wiederholung Zeichenkette

Karlsruher Institut für Technologie

String

- Zusammensetzung von chars
- Ist kein primitiver Datentyp, sondern eine Klasse
- Außerdem sind ein paar Besonderheiten z.B. beim Vergleichen zu beachten
- Operation: Konkatenation

String
- Zusammensetzung von chars
- Ist kein primitiver Datentyp, sondern eine Klasse
- Außerdem sind ein paar Besonderheiten z.B. beim Vergleichen zu beachten
- Operation: Konkatenation

Konkatenation

Strings werden mit dem + Operator konkateniert (verkettet):

```
1 String s1 = "Dies ist ";
2 String s2 = "eine Konkatenation";
3 String s3 = s1 + s2 + "!";
4 String s4 = "Dies ist + keine Konkatenation!";
5 System.out.println(s3);
6 System.out.println(s4);
```

Ausgabe:

Strings werden mit dem + Operator konkateniert (verkettet):

```
1 String s1 = "Dies ist ";
2 String s2 = "eine Konkatenation";
3 String s3 = s1 + s2 + "!";
4 String s4 = "Dies ist + keine Konkatenation!";
5 System.out.println(s3);
6 System.out.println(s4);
```

Ausgabe:

Konkatenation

Strings werden mit dem + Operator konkateniert (verkettet):

```
1 String s1 = "Dies ist ";
2 String s2 = "eine Konkatenation";
3 String s3 = s1 + s2 + "!";
4 String s4 = "Dies ist + keine Konkatenation!";
5 System.out.println(s3);
6 System.out.println(s4);
```

Ausgabe:

```
Dies ist eine Konkatenation!
Dies ist + keine Konkatenation!
```


Strings werden mit dem + Operator konkateniert (verkettet):

```
1 String s1 = "Dies ist ";
2 String s2 = "eine Konkatenation";
3 String s3 = s1 + s2 + "!";
4 String s4 = "Dies ist + keine Konkatenation!";
5 System.out.println(s3);
6 System.out.println(s4);
```

Ausgabe:

```
Dies ist eine Konkatenation!
Dies ist + keine Konkatenation!
```

Übungen

Karlsruher Institut für Technologie

Was wird Ausgegeben?

```
class References {
    public static void main(String[] args) {
        String s1 = "Vorsicht mit ";
        String s2 = s1;
        s2 = s2 + "Strings";
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
    }
}
```


Übungen

Karlsruher Institut für Technologie

Was wird Ausgegeben?

```java
class References {
    public static void main(String[] args) {
        String s1 = "Vorsicht mit ";
        String s2 = s1;
        s2 = s2 + "Strings";
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
    }
}
```

```
$ java References
s1: Vorsicht mit
s2: Vorsicht mit Strings
```


Warum ist das so?

Konkatenation mit +

- Bei der Nutzung von + wird ein neues Objekt erstellt
- Die Referenz zu „s1“ wird dabei gelöscht
- Deshalb ändert sich nur „s2“

Warum ist das so?

Konkatenation mit +

- Bei der Nutzung von + wird ein neues Objekt erstellt
- Die Referenz zu „s1“ wird dabei gelöscht
- Deshalb ändert sich nur „s2“

`immutable`

- String ist `immutable`, also unveränderbar

Übungen

Karlsruher Institut für Technologie

Was wird Ausgegeben?

```java
class References2 {
    public static void main(String[] args) {
        String s1 = "Vorsicht mit ";
        String s2 = s1;
        s2.concat("Strings");
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
    }
}
```

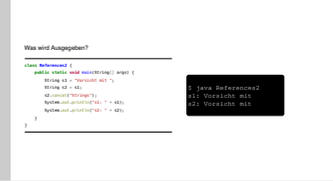

Übungen

Karlsruher Institut für Technologie

Was wird Ausgegeben?

```java
class References2 {
    public static void main(String[] args) {
        String s1 = "Vorsicht mit ";
        String s2 = s1;
        s2.concat("Strings");
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
    }
}
```

```
$ java References2
s1: Vorsicht mit
s2: Vorsicht mit
```

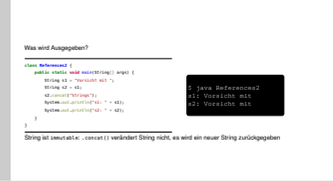
Übungen

Was wird Ausgegeben?

```java
class References2 {
    public static void main(String[] args) {
        String s1 = "Vorsicht mit ";
        String s2 = s1;
        s2.concat("Strings");
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
    }
}
```

```
$ java References2
s1: Vorsicht mit
s2: Vorsicht mit
```

String ist `immutable`: `.concat()` verändert String nicht, es wird ein neuer String zurückgegeben



Strings vergleichen

Vergleiche von primitiven Datentypen (==, >=, ...)

- Können direkt verglichen werden
- Es wird direkt der im Speicher hinterlegte Wert verglichen

Strings vergleichen

Vergleiche von primitiven Datentypen (==, >=, ...)

- Können direkt verglichen werden
- Es wird direkt der im Speicher hinterlegte Wert verglichen

Vergleiche von Strings

- == vergleicht nur die im Speicher hinterlegten Werte beider Variablen
- Bei Objekten werden also nur die Referenzen verglichen
- Attributwerte werden ignoriert

Strings vergleichen

Vergleiche von primitiven Datentypen (==, >=, ...)

- Können direkt verglichen werden
- Es wird direkt der im Speicher hinterlegte Wert verglichen

Vergleiche von Strings

- == vergleicht nur die im Speicher hinterlegten Werte beider Variablen
- Bei Objekten werden also nur die Referenzen verglichen
- Attributwerte werden ignoriert
- Benutzung der Funktion `equals(String string)`
- Für Klasse `String` ist `equals` bereits implementiert
- Später: Nutzung von `equals` für eigene Objekte

Anwendung

```java
class StringComparator {
    public static void main(String[] args) {
        String string1 = "string";
        String string2 = new String("string");
        System.out.println(string1 == string2);
        System.out.println(string1.equals(string2));
    }
}
```


Anwendung

```java
class StringComparator {
    public static void main(String[] args) {
        String string1 = "string";
        String string2 = new String("string");
        System.out.println(string1 == string2);
        System.out.println(string1.equals(string2));
    }
}
```

```
$ java StringComparator
false
true
```


Konstruktoren - Einführung

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

1. parameter ist der aussagekräftiger Bezeichner des übergebenen Arguments
2. Parameter-List kann leer sein: keine Argumente werden übergeben

Konstruktoren - Einführung

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

Schema:

```
KlassenName(Parameter-Liste) {
    // Konstruktor-Rumpf
}
```

Parameter-Liste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten
Schema:
KlassenName(Parameter-Liste) {
// Konstruktor-Rumpf
}
Parameter-Liste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

1. parameter ist der aussagekräftiger Bezeichner des übergebenen Arguments
2. Parameter-List kann leer sein: keine Argumente werden übergeben

Konstruktoren - Einführung

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

Schema:

```
KlassenName(Parameter-Liste) {
    // Konstruktor-Rumpf
}
```

Parameter-Liste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

Aufruf: `new KlassenName(aktuelle Parameter);`

- Dienen der Initialisierung eines Objekts (Setzen von Anfangswerten der Attribute)
- legen damit den Startzustand eines Objekts fest
- Können Anfangswerte des Objekts als Parameter erhalten

Schema:

```
KlassenName(Parameter-Liste) {
    // Konstruktor-Rumpf
}
```

Parameter-Liste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

Aufruf: `new KlassenName(aktuelle Parameter);`

1. parameter ist der aussagekräftiger Bezeichner des übergebenen Arguments
2. Parameter-List kann leer sein: keine Argumente werden übergeben

Konstruktor - Beispiel

Karlsruher Institut für Technologie

Bisher:

```java
class Person {
    String name;
    int age;
}

Person person = new Person();
person.name = "Hans Peter";
person.age = 68;
```


Konstruktor - Beispiel

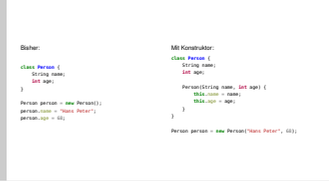
Karlsruher Institut für Technologie

Bisher:

```
class Person {
    String name;
    int age;
}

Person person = new Person();
person.name = "Hans Peter";
person.age = 68;
```

Mit Konstruktor:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

Person person = new Person("Hans Peter", 68);
```

Konstruktor - Beispiel

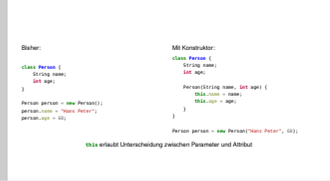
Bisher:

```java
class Person {
    String name;
    int age;
}

Person person = new Person();
person.name = "Hans Peter";
person.age = 68;
```

Mit Konstruktor:

```java
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

Person person = new Person("Hans Peter", 68);
```

`this` erlaubt Unterscheidung zwischen Parameter und Attribut



Mehrere Konstruktoren

- festlegen mehrer Konstuktoren möglich
- müssen unterschiedliche Parameteranzahl und/oder unterschiedliche Parametertypen habe

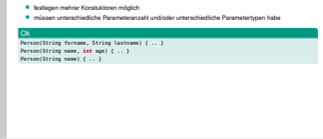
- festlegen mehrer Konstuktoren möglich
- müssen unterschiedliche Parameteranzahl und/oder unterschiedliche Parametertypen habe

Mehrere Konstruktoren

- festlegen mehrer Konstuktoren möglich
- müssen unterschiedliche Parameteranzahl und/oder unterschiedliche Parametertypen habe

Ok

```
Person(String forname, String lastname) { .. }
Person(String name, int age) { .. }
Person(String name) { .. }
```

Mehrere Konstruktoren

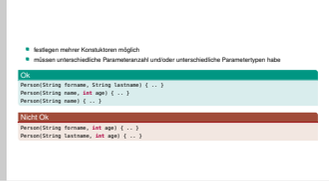
- festlegen mehrer Konstuktoren möglich
- müssen unterschiedliche Parameteranzahl und/oder unterschiedliche Parametertypen habe

Ok

```
Person(String forname, String lastname) { .. }
Person(String name, int age) { .. }
Person(String name) { .. }
```

Nicht Ok

```
Person(String forname, int age) { .. }
Person(String lastname, int age) { .. }
```

Mehrere Konstruktoren - Beispiel

Intuition:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    Person(String name) {
        this.name = name;
        this.age = 0;
    }
}
```


Mehrere Konstruktoren - Beispiel

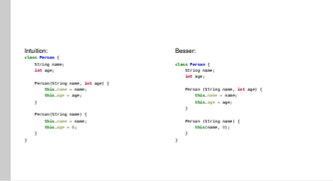
Karlsruher Institut für Technologie

Intuition:

```java
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    Person(String name) {
        this.name = name;
        this.age = 0;
    }
}
```

Besser:

```java
class Person {
    String name;
    int age;

    Person (String name, int age) {
        this.name = name;
        this.age = age;
    }

    Person (String name) {
        this(name, 0);
    }
}
```

Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`

Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte

Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte
- sobald ein Konstuktor angegeben wird, verschwindet der Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte
- sobald ein Konstuktor angegeben wird, verschwindet der Default-Konstruktor

Default-Konstruktor

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte
- sobald ein Konstuktor angegeben wird, verschwindet der Default-Konstruktor

Typ	Default-Werte
boolean	false
byte, short, int	0
long	0L
float	0.0f
double	0.0
char	'\u0000'
Objekt-Referenz	null

- ist kein Konstruktor angegeben, wird ein leerer Konstuktor automatisch ergänzt
- `KlassenName() { }`
- dieser initialisiert Attribute auf ihre Default-Werte
- sobald ein Konstuktor angegeben wird, verschwindet der Default-Konstruktor

Typ	Default-Werte
boolean	false
byte, short, int	0
long	0L
float	0.0f
double	0.0
char	'\u0000'
Objekt-Referenz	null

Methoden - Einführung

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

1. paramter ist der aussagekräftiger Bezeichner des übergebens Arguments

Methoden - Einführung

Karlsruher Institut für Technologie

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```


Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

1. paramter ist der aussagekräftiger Bezeichner des übergebens Arguments

Methoden - Einführung

Karlsruher Institut für Technologie

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

Der Rückgabetyp **`void`** steht dabei für keine Rückgabe!

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

Der Rückgabetyp **`void`** steht dabei für keine Rückgabe!

1. paramter ist der aussagekräftiger Bezeichner des übergebens Arguments

Methoden - Einführung

Karlsruher Institut für Technologie

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

Der Rückgabetyp **`void`** steht dabei für keine Rückgabe!

- Parameterliste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

Methoden realisieren das dynamische Verhalten von Objekten und führen Berechnungen durch.

Schema:

```
Rückgabetyp methodenName(Parameterliste) {
    // Methodenrumpf
}
```

Der Rückgabetyp **`void`** steht dabei für keine Rückgabe!

- Parameterliste: Datentyp$_1$ parameter$_1$, Datentyp$_2$ parameter$_2$, ..., Datentyp$_n$ parameter$_n$

1. paramter ist der aussagekräftiger Bezeichner des übergebens Arguments

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.
Die Methodensignatur besteht aus:

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:
- dem Namen der Methode

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode
- der Anzahl der Parameter

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:
- dem Namen der Methode
- der Anzahl der Parameter

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:
- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter
- der Typen der Parameter

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:
- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter
- der Typen der Parameter

Methoden - Aufbau

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:

- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter
- der Typen der Parameter

$$\underbrace{\text{Rückgabetyp}\ \underbrace{\text{methodenName(Parameterliste)}}_{\text{Methodensignatur}}\{\ \ldots\ \}}_{\text{Methodenkopf}}$$

Der Methodenkopf besteht aus dem Rückgabetyp und der Methodensignatur.

Die Methodensignatur besteht aus:
- dem Namen der Methode
- der Anzahl der Parameter
- der Reihenfolge der Parameter
- der Typen der Parameter

Rückgabetyp methodenName(Parameterliste){ ... }
Methodensignatur
Methodenkopf

Methoden - Zugriff auf Attribute

In Methoden kann auch auf Attribute des Objekts zugeriffen werden:

In Methoden kann auch auf Attribute des Objekts zugeriffen werden:

Methoden - Zugriff auf Attribute

In Methoden kann auch auf Attribute des Objekts zugeriffen werden:

```java
class Person {
    String name = "Sven";

    void greet(String name) {
        System.out.println("Hallo " + name + ", ich heiße " + this.name + ".");
    }
}
```


In Methoden kann auch auf Attribute des Objekts zugeriffen werden:
class Person {
String name = "Sven";
void greet(String name) {
System.out.println("Hallo " + name + ", ich heiße " + this.name + ".");
}
}

Methoden - Zugriff auf Attribute

In Methoden kann auch auf Attribute des Objekts zugeriffen werden:

```
class Person {
    String name = "Sven";

    void greet(String name) {
        System.out.println("Hallo " + name + ", ich heiße " + this.name + ".");
    }
}
```

`this` erlaubt Unterscheidung zwischen Parameter und Attribut

In Methoden kann auch auf Attribute des Objekts zugeriffen werden:
class Person {
String name = "Sven";
void greet(String name) {
System.out.println("Hallo " + name + ", ich heiße " + this.name + ".");
}
}
this erlaubt Unterscheidung zwischen Parameter und Attribut

Methoden - Aufruf

- Methoden können nur auf Objekten aufgerufen werden!

- Methoden können nur auf Objekten aufgerufen werden!

1. Bei Aufruf in selber Klasse nicht nötig: implizit this genutzt

Methoden - Aufruf

- Methoden können nur auf Objekten aufgerufen werden!
  - Dazu wird die Syntax `objekt.methode()` verwendet.

1. Bei Aufruf in selber Klasse nicht nötig: implizit this genutzt

Methoden - Aufruf

- Methoden können nur auf Objekten aufgerufen werden!
  - Dazu wird die Syntax `objekt.methode()` verwendet.

Beispiel

```
Person person = new Person();
person.greet("Peter");
```


1. Bei Aufruf in selber Klasse nicht nötig: implizit this genutzt

Methoden - Aufruf

- Methoden können nur auf Objekten aufgerufen werden!
  - Dazu wird die Syntax `objekt.methode()` verwendet.

Beispiel

```
Person person = new Person();
person.greet("Peter");
```

Ausgabe:

```
Hallo Peter, ich heiße Sven.
```


1. Bei Aufruf in selber Klasse nicht nötig: implizit this genutzt

Methoden - Rückgabe

Karlsruher Institut für Technologie

Wird ein anderer Rückgabetyp als **`void`** gewählt, muss die Methode einen Wert vom Rückgabetyp mit dem Schlüsselwort **`return`** zurückgeben.

Wird ein anderer Rückgabetyp als void gewählt, muss die Methode einen Wert vom Rückgabetyp mit dem Schlüsselwort return zurückgeben.

Methoden - Rückgabe

Wird ein anderer Rückgabetyp als **`void`** gewählt, muss die Methode einen Wert vom Rückgabetyp mit dem Schlüsselwort **`return`** zurückgeben.

Beispiel

```
int sum(int a, int b) {
    return a + b;
}
```


Methoden - return bei void

`return` kann aber auch bei Methoden mit Rückgabetyp `void` benutzt werden:
z.B. um eine Methode gezielt zu verlassen

return kann aber auch bei Methoden mit Rückgabetyp void benutzt werden:
z.B. um eine Methode gezielt zu verlassen

1. Geschweifte Klammern beim if setzten. Nur für auf Folien oke

Methoden - return bei void

`return` kann aber auch bei Methoden mit Rückgabetyp `void` benutzt werden:
z.B. um eine Methode gezielt zu verlassen

Beispiel

```
class Job {
    boolean started = false;
    void startJob() {
        if (started)
            return;
        started = true;
        // Do the Job
    }
}
```


return kann aber auch bei Methoden mit Rückgabetyp void benutzt werden:
z.B. um eine Methode gezielt zu verlassen
Beispiel
class Job {
boolean started = false;
void startJob() {
if (started)
return;
started = true;
// Do the Job
}
}

1. Geschweifte Klammern beim if setzten. Nur für auf Folien oke

Klassenmethoden

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
```


Methoden die mit dem Modifier static beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.
class Math {
static int sum(int a, int b) {
return a + b;
}
}

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
```


Methoden die mit dem Modifier static beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.
class Math {
static int sum(int a, int b) {
return a + b;
}
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier **`static`** beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```java
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
```

Es wird **kein** Objekt benötigt, um die Methode auszuführen.

Methoden die mit dem Modifier `static` beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```java
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
```

Es wird **kein** Objekt benötigt, um die Methode auszuführen.

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier **`static`** beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
```

Es wird **kein** Objekt benötigt, um die Methode auszuführen.
Schema: Klasse.methodenName();

Methoden die mit dem Modifier static beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.
class Math {
static int sum(int a, int b) {
return a + b;
}
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
Es wird kein Objekt benötigt, um die Methode auszuführen.
Schema: Klasse.methodenName();

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Klassenmethoden

Methoden die mit dem Modifier **`static`** beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.

```
class Math {
    static int sum(int a, int b) {
        return a + b;
    }
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
```

Es wird **kein** Objekt benötigt, um die Methode auszuführen.
Schema: Klasse.methodenName();

`this` kann in Klassenmethoden nicht genutzt werden

Methoden die mit dem Modifier static beginnen nennt man Klassenmethoden.
Diese gehören zu der Klasse statt zu einem Objekt.
class Math {
static int sum(int a, int b) {
return a + b;
}
}
int c = Math.sum(5, 8); // Nach Ausfühung: c = 13
Es wird kein Objekt benötigt, um die Methode auszuführen.
Schema: Klasse.methodenName();
this kann in Klassenmethoden nicht genutzt werden

1. Mit static kann man auch Attribute als Klassenvariable definieren
2. Zugriff auf diese Analog zum Methode: Klasse.variable

Hilfsmethoden

Karlsruher Institut für Technologie

Was sind Hilfsmethoden?

- Methoden, die häufig verwendete Funktionalität ausführen
- Können statisch oder nicht statisch sein

Was sind Hilfsmethoden?
- Methoden, die häufig verwendete Funktionalität ausführen
- Können statisch oder nicht statisch sein

Hilfsmethoden

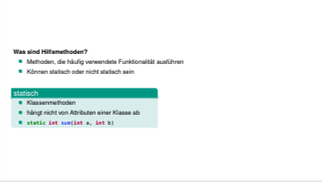
Karlsruher Institut für Technologie

Was sind Hilfsmethoden?

- Methoden, die häufig verwendete Funktionalität ausführen
- Können statisch oder nicht statisch sein

statisch

- Klassenmethoden
- hängt nicht von Attributen einer Klasse ab
- `static int sum(int a, int b)`



Hilfsmethoden

Karlsruher Institut für Technologie

Was sind Hilfsmethoden?

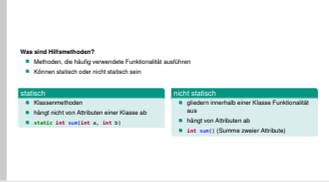
- Methoden, die häufig verwendete Funktionalität ausführen
- Können statisch oder nicht statisch sein

statisch

- Klassenmethoden
- hängt nicht von Attributen einer Klasse ab
- `static int sum(int a, int b)`

nicht statisch

- gliedern innerhalb einer Klasse Funktionalität aus
- hängt von Attributen ab
- `int sum()` (Summe zweier Attribute)



Überladen

Es können mehrere Methoden mit dem gleichen Namen existieren, wenn sich diese in den Datentypen, Reihenfolge und/oder der Anzahl ihrer Parameter(d.h. in ihrer Signatur) unterscheiden.

Ok

```
int sum (int a, int b) { ... }
double sum (double a, double b) { ... }
int sum (int a, int b, int c) { ... }
```

Nicht Ok

```
int sum (int a, int b) { ... }
long sum (int a, int b) { ... }
int sum (int a, int c) { ... }
```

Methoden gleichen Namens, die sich nur im Rückgabetyp unterscheiden, sind nicht möglich.

Methoden - Übung

Aufgabe

Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

Aufgabe
Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

am besten die Aufgabe live vormachen. Input der Tutanden in Eclipse umsetzen. Danach MusterlösungMethode muss nicht unbedingt static sein

Methoden - Übung

Aufgabe

Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

Tipp

1. Mit dem Modulo Operator % erhaltet ihr den Rest der Ganzzahldivision.

am besten die Aufgabe live vormachen. Input der Tutanden in Eclipse umsetzen. Danach MusterlösungMethode muss nicht unbedingt static sein

Methoden - Übung

Aufgabe

Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

Tipp

1. Mit dem Modulo Operator % erhaltet ihr den Rest der Ganzzahldivision.
2. Der Rückgabetyp euer Methode muss `boolean` sein.

am besten die Aufgabe live vormachen. Input der Tutanden in Eclipse umsetzen. Danach MusterlösungMethode muss nicht unbedingt static sein

Methoden - Übung

Aufgabe

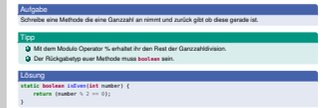
Schreibe eine Methode die eine Ganzzahl an nimmt und zurück gibt ob diese gerade ist.

Tipp

1. Mit dem Modulo Operator % erhaltet ihr den Rest der Ganzzahldivision.
2. Der Rückgabetyp euer Methode muss **`boolean`** sein.

Lösung

```
static boolean isEven(int number) {
    return (number % 2 == 0);
}
```

am besten die Aufgabe live vormachen. Input der Tutanden in Eclipse umsetzen. Danach MusterlösungMethode muss nicht unbedingt static sein

Methoden - JavaDoc

- beschreibt Zweck der Methode

beschreibt Zweck der Methode

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)
- der Rückgabetyp wird beschrieben (Inhalt)

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)
- der Rückgabetyp wird beschrieben (Inhalt)

1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

- beschreibt Zweck der Methode
- beschreibt in welchem Kontext die Methode verwendet werden kann
  - Vor- und Nachbedingungen der Ausführung
  - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und Wertebereich)
- der Rückgabetyp wird beschrieben (Inhalt)

Syntax

```
/**
 * Kurze aber aussagekräftige Beschreibung von Zweck und Kontext der Methode.
 * @param parameter Beschreibung des Parameters
 * @return Beschreibung Rückgabewert
 */
RückgabeDatentyp methodenName(DatenTyp parameter) {
```


1. Beispiel Spezialfälle: Wurzel ziehen, Methode verarbeitet keine negativen werte: muss angegeben werden

Methoden - JavaDoc

Karlsruher Institut für Technologie

```java
/**
 * Returns the absolute value of an int value.
 * If the argument is not negative, the argument is returned.
 * If the argument is negative, the negation of the argument is returned.
 *
 * Note that if the argument is equal to the value of Integer.MIN_VALUE,
 * the most negative representable int value, the result is that same value, which is negative.
 *
 * @param a the argument whose absolute value is to be determined
 * @return the absolute value of the argument.
 */
static int abs(int a) {
    return (a < 0) ? -a : a;
}
```


/**
 * Returns the absolute value of an int value.
 * If the argument is not negative, the argument is returned.
 * If the argument is negative, the negation of the argument is returned.
 *
 * Note that if the argument is equal to the value of Integer.MIN_VALUE,
 * the most negative representable int value, the result is that same value, which is negative.
 *
 * @param a the argument whose absolute value is to be determined
 * @return the absolute value of the argument.
 */
static int abs(int a) {
    return (a < 0) ? -a : a;
}

Sauber Programmieren

Negativ Beispiel

```
public class DemoBad{public static void main(String[]
args){System.out.println("Hello world!");System.
out.println("auto-generated text");}}
```


Sauber Programmieren

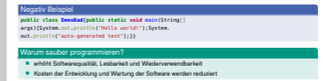
Karlsruher Institut für Technologie

Negativ Beispiel

```
public class DemoBad{public static void main(String[]
args){System.out.println("Hello world!");System.
out.println("auto-generated text");}}
```

Warum sauber programmieren?

- erhöht Softwarequalität, Lesbarkeit und Wiederverwendbarkeit
- Kosten der Entwicklung und Wartung der Software werden reduziert



Sauber Programmieren

Aber wie?

- Namen und Bezeichner sind aussagekräftig und in Englisch (Bennenungskonventionen beachten)
- Kommentare existieren und sind sinnvoll (JavaDoc)
- Einheitliche Sprache in Kommentaren (nur Englisch oder Deutsch)
- Quelltext enthält sinnvolle Einrückungen und Leerzeichen

Aber wie?
- Namen und Bezeichner sind aussagekräftig und in Englisch (Bennenungskonventionen beachten)
- Kommentare existieren und sind sinnvoll (JavaDoc)
- Einheitliche Sprache in Kommentaren (nur Englisch oder Deutsch)
- Quelltext enthält sinnvolle Einrückungen und Leerzeichen

Sauber Programmieren

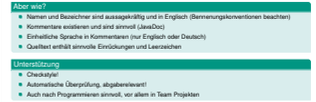
Karlsruher Institut für Technologie

Aber wie?

- Namen und Bezeichner sind aussagekräftig und in Englisch (Bennenungskonventionen beachten)
- Kommentare existieren und sind sinnvoll (JavaDoc)
- Einheitliche Sprache in Kommentaren (nur Englisch oder Deutsch)
- Quelltext enthält sinnvolle Einrückungen und Leerzeichen

Unterstützung

- Checkstyle!
- Automatische Überprüfung, abgaberelevant!
- Auch nach Programmieren sinnvoll, vor allem in Team Projekten



Sauber Programmieren - Beispiel

Karlsruher Institut für Technologie

```java
/**
 * The class DemoNice looks way better!
 * @author Gregor Lucka
 * @see DemoBad
 */
public class DemoNice {
    /**
     * This main-method greets the world.
     * @params args This methods does not need any arguments.
     */
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```


Bis zum nächsten Tutorium am 23.11.2022!

Bis zum nächsten Tutorium am 23.11.2022!