

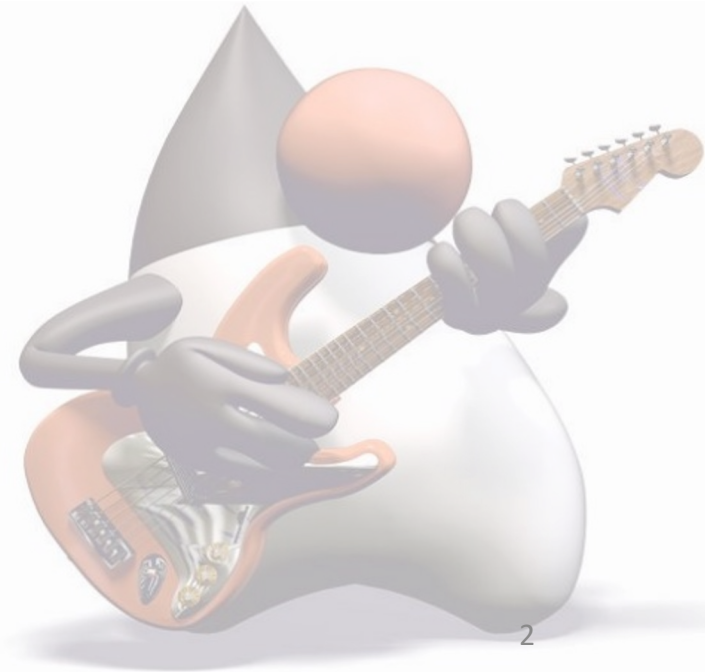
Java Fundamentals

Komplexe Klassen



Übergabe von Primitiven Typen

```
public class Caller {  
    public void calculate(){  
        int a = 5;  
        for (int i=0;i<10;i++){  
            add(a);  
        }  
        System.out.println("result = " + a);  
    }  
  
    public void add(int b){  
        b = b + 5;  
    }  
  
    public static void main(String[] args){  
        Caller caller = new Caller();  
        caller.calculate();  
    }  
}
```



Übergabe von Primitiven Typen

```
public class Caller {  
    public void calculate(){  
        int a = 5;  
        add(a);  
        System.out.println("result = " + a);  
    }  
  
    public void add(int b){  
        b = b + 5;  
    }  
  
    public static void main(String[] args){  
        Caller caller = new Caller();  
        caller.calculate();  
    }  
}
```

Ergebnis = 5

a 5

b 5

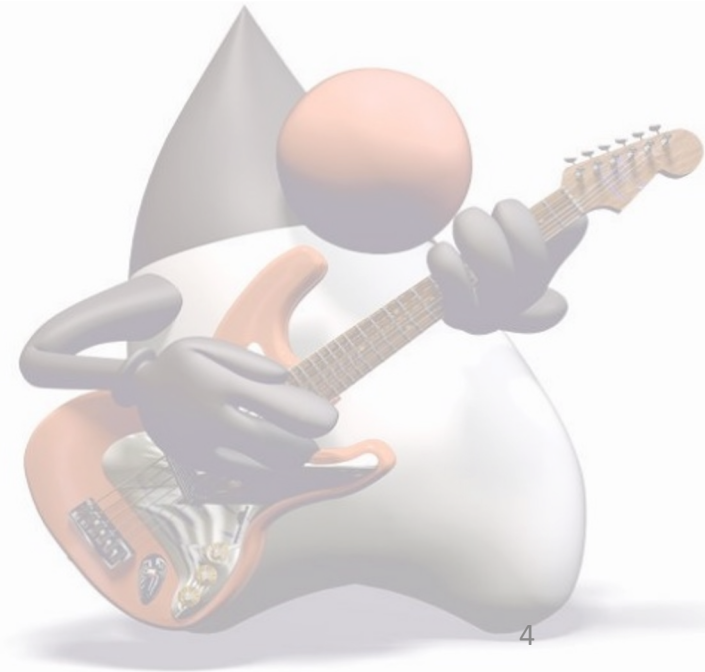
b 10



Übergabe von Objekten

```
public class Caller2 {  
    public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
  
        System.out.println("result = " + a.value);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller caller = new Caller();  
        caller.calculate();  
    }  
}  
  
public class Number{  
    public int value;  
}
```

20.09.22



Übergabe von Referenz Typen

```
public class Caller2 {  
    public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
  
        System.out.println("result = " + a);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller caller = new Caller();  
        caller.calculate();  
    }  
}
```

```
public class Number{  
    public int value;  
}
```

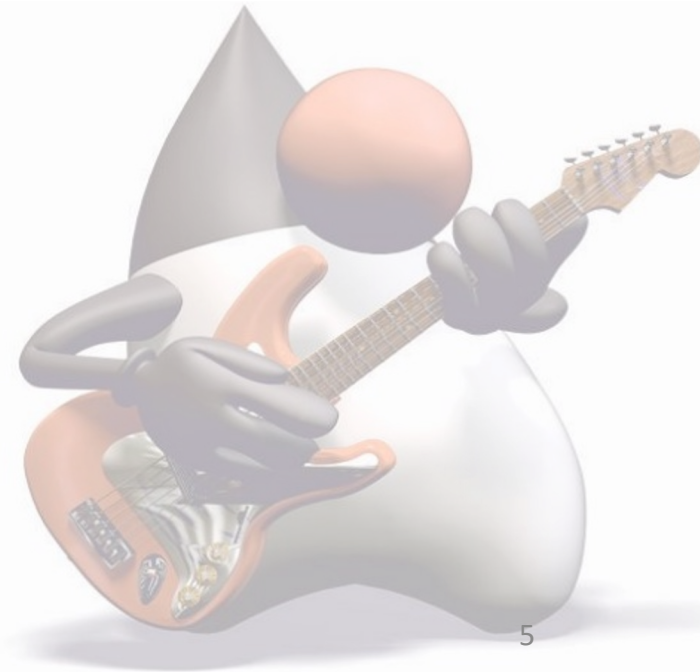
a 0x12ff

copy

b 0x12ff

:Number
a=5

Ergebnis = 10

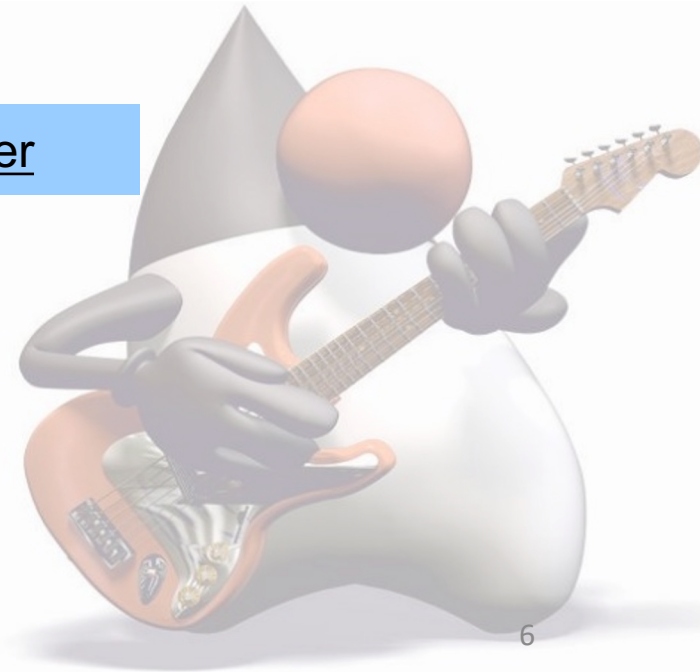


Garbage Collection

```
public class Caller2 {  
    public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
  
        System.out.println("result = " + a);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller caller = new Caller();  
        caller.calculate();  
        System.out.println("finished");  
        for (int i = 0; i < 10000; i++) {  
            i = i + 1;  
        }  
    }  
}
```

➔

:Caller



Garbage Collection

```
public class Caller2 {  
    public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
  
        System.out.println("result = " + a);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller caller = new Caller();  
        caller.calculate();  
        System.out.println("finished");  
        for (int i = 0; i < 10000; i++) {  
            i = i + 1;  
        }  
    }  
}
```

:Caller



Garbage Collection

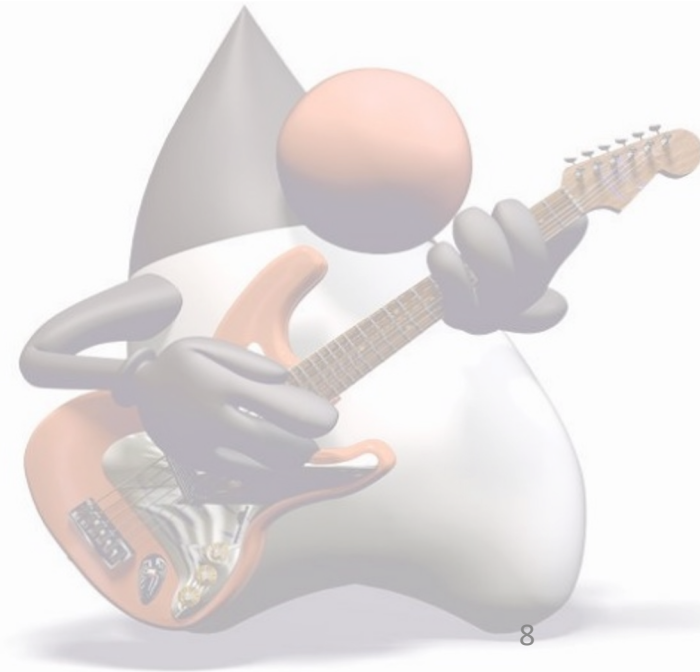
```
public class Caller2 {  
    → public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
        System.out.println("result = " + a);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller2 caller = new Caller2();  
        caller.calculate();  
        System.out.println("finished");  
        for (int i = 0; i < 10000; i++) {  
            i = i + 1;  
        }  
    }  
}
```

:Number

:Caller2

}

20.09.22

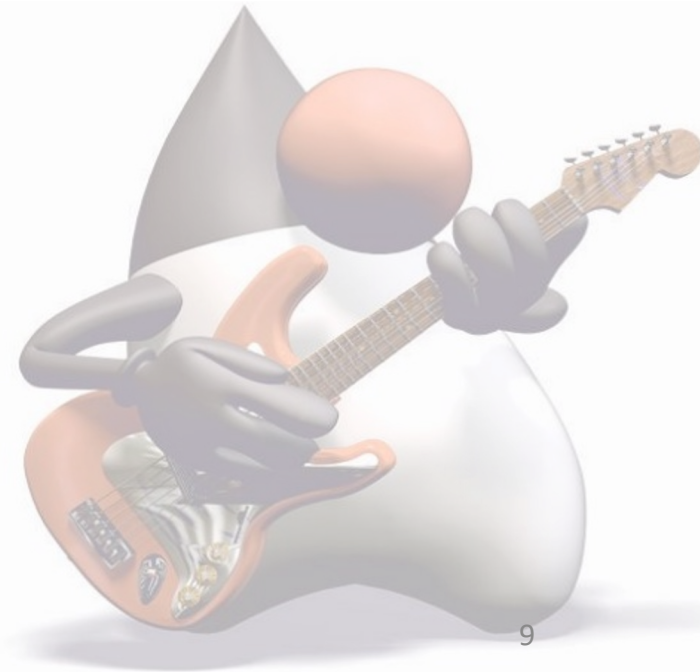


Garbage Collection

```
public class Caller2 {  
    public void calculate() {  
        Number a = new Number();  
        a.value = 5;  
        add(a);  
  
        System.out.println("result = " + a);  
    }  
  
    public void add(Number b) {  
        b.value = b.value + 5;  
    }  
  
    public static void main(String[] args) {  
        Caller2 caller = new Caller2();  
        caller.calculate();  
        System.out.println("finished");  
        for (int i = 0; i < 10000; i++) {  
            i = i + 1;  
        }  
    }  
}
```

~~:Number~~

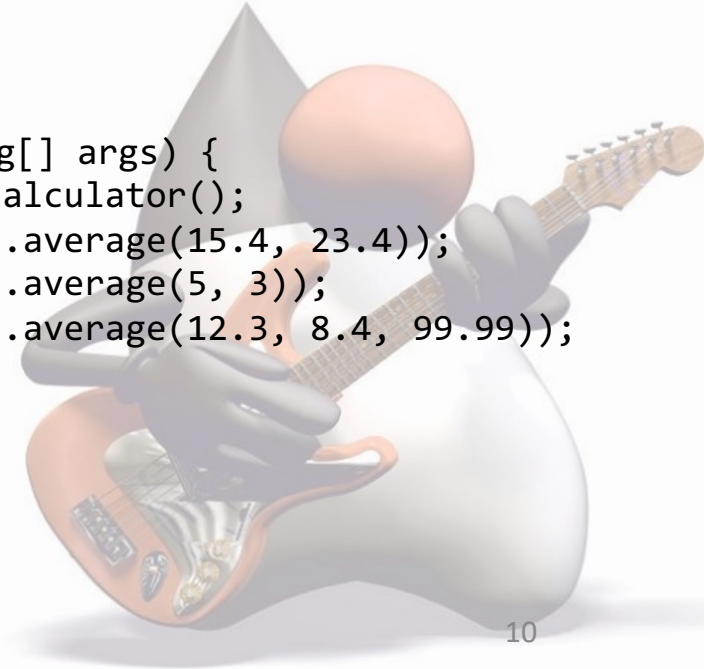
:Caller2



Overloading

- Es können gleichnamige Methoden mit unterschiedlichen Parametern (Typen und/oder Anzahl) definiert werden
- Das wird als Überladen (Overloading) von Methoden bezeichnet.

```
public class Calculator {  
    public int average(int a, int b) {  
        int result = (a + b) / 2;  
        return result;  
    }  
  
    public double average(double a, double b) {  
        double result = (a + b) / 2.;  
        return result;  
    }  
  
    public double average(double a, double b, double c) {  
        double result = (a + b + c) / 3.;  
        return result;  
    }  
  
    public static void main(String[] args) {  
        Calculator c = new Calculator();  
        System.out.println(c.average(15.4, 23.4));  
        System.out.println(c.average(5, 3));  
        System.out.println(c.average(12.3, 8.4, 99.99));  
    }  
}
```



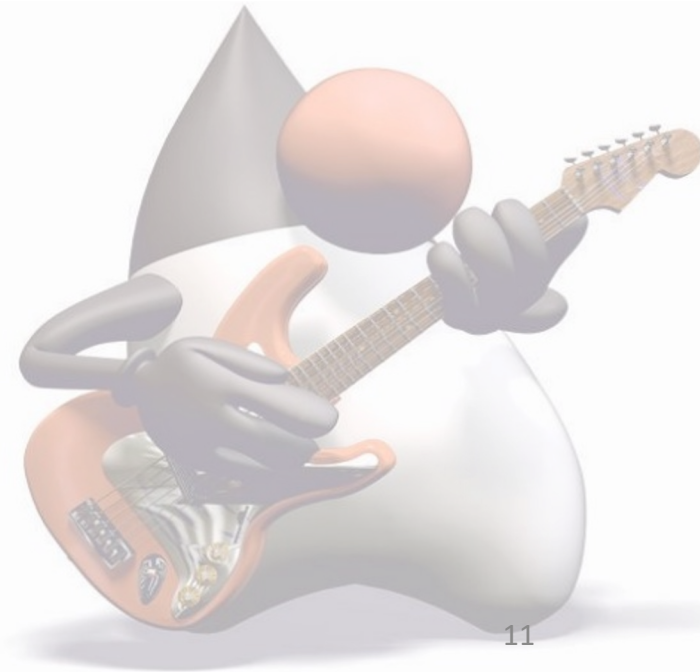
Constructors

Ein Konstruktor ist eine Methode, die aufgerufen wird wenn ein Objekt initialisiert wird.

```
public class Customer {  
    public String name = "";  
    public float height = 0.0F;  
    public float weight = 0.0F;  
  
    public Customer() {  
        height = 1.75F;  
        weight = 120.F;  
        name = "Anonymous";  
    }  
    //...  
}
```

1. execute explicit
initializers

2. execute constructor

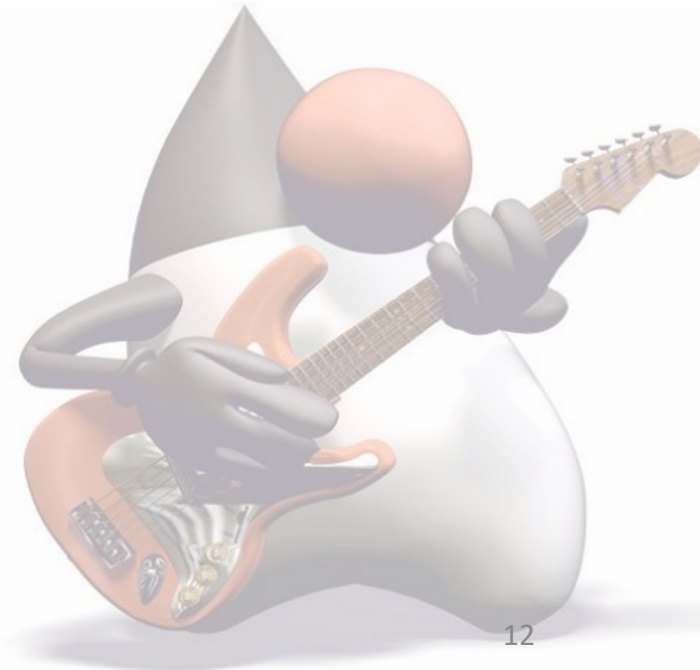


Constructors

```
public class Customer {  
    public String name = "";  
    public float height = 0.0F;  
    public float weight = 0.0F;  
  
    //...  
}
```

```
public class Customer {  
    public String name = "";  
    public float height = 0.0F;  
    public float weight = 0.0F;  
  
    public Customer() {  
    }  
  
    //...  
}
```

Jede Klasse hat einen Konstruktor, wenn man keinen eigenen Konstruktor definiert dann wird ein Default Konstruktor generiert.

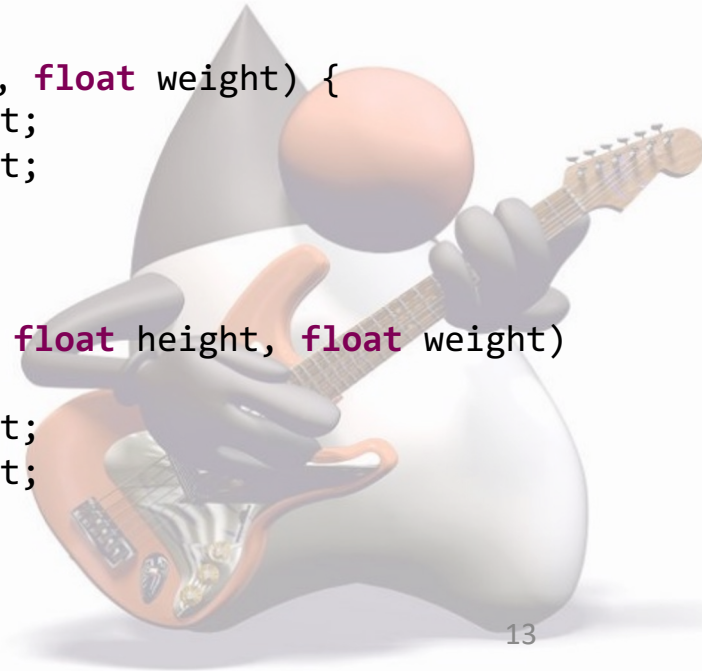


Constructors

- Eine Klasse kann mehrere überladene Konstruktoren haben.
- Sobald man einen Konstruktor definiert hat, wird kein „default“ Konstruktor mehr generiert.

```
public class Customer {  
    public String name = "";  
    public float height = 0.0F;  
    public float weight = 0.0F;  
  
    public Customer() {  
        height = 1.75F;  
        weight = 120.F;  
        name = "Anonymous";  
    }  
  
    public Customer(float height, float weight) {  
        this.height = height;  
        this.weight = weight;  
        name = "Anonymous";  
    }  
  
    public Customer(String name, float height, float weight)  
    {  
        this.height = height;  
        this.weight = weight;  
        this.name = name;  
    }  
}
```

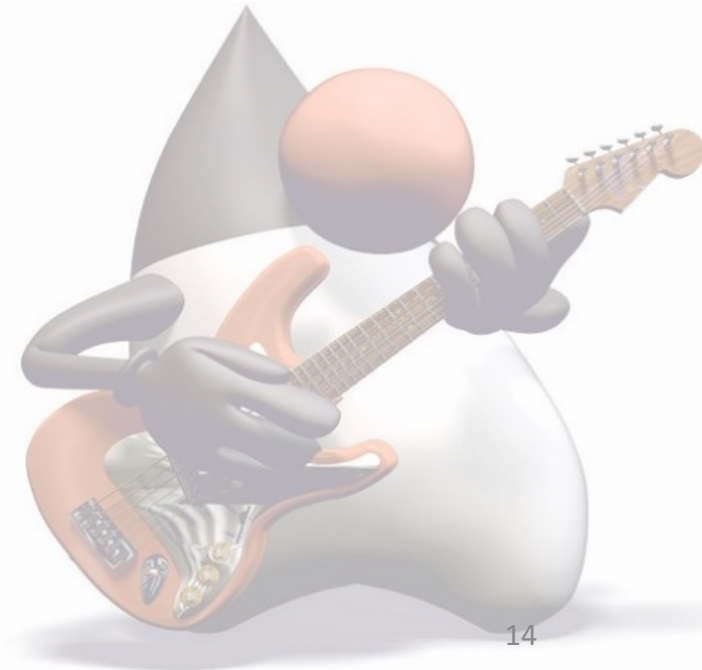
...



Verkettete Konstruktoren

```
public class Customer {  
    public String name = "";  
    public float height = 0.0F;  
    public float weight = 0.0F;  
  
    public Customer() {  
        this("Anonymous", 1.75F, 120.F);  
    }  
  
    public Customer(float height, float weight) {  
        this("Anonymous", height, weight);  
    }  
  
    public Customer(String name, float height, float weight)  
    {  
        this.height = height;  
        this.weight = weight;  
        this.name = name;  
    }  
    ...  
}
```

- Konstruktoren können sich gegenseitig über `this(params..)` aufrufen.
- `this(...)` muss das erste Statement im Konstruktor sein.



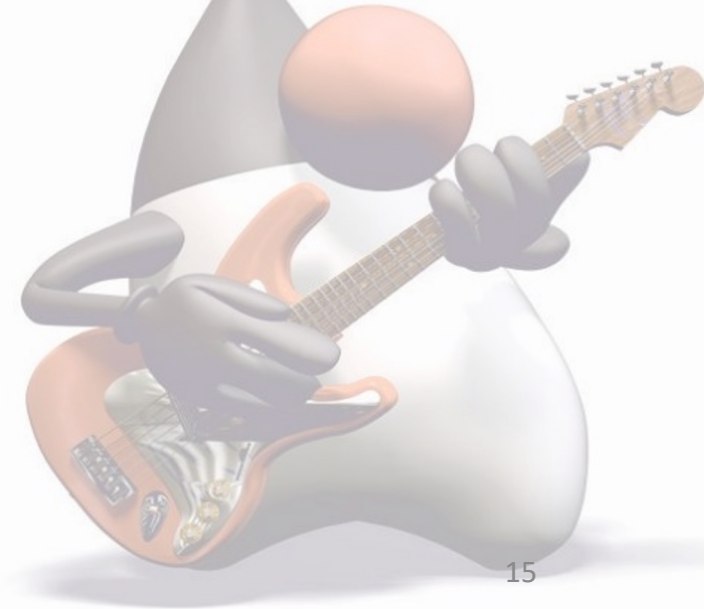
Objekte in Strings konvertieren

- Die JVM verwendet die toString() Methode um Objekte in Strings umzuwandeln
- Die „Default“ Implementierung der toString() Methode gibt den Klassennamen + Referenzadresse aus.

```
package at.java;
```

```
public class Bank {  
    public String name = "Bank Austria";  
    public String address = "1010 Wien";  
  
    public static void main(String[] args) {  
        Bank bank = new Bank();  
        System.out.println(bank);  
    }  
}
```

```
at.java.Bank@5224ee
```

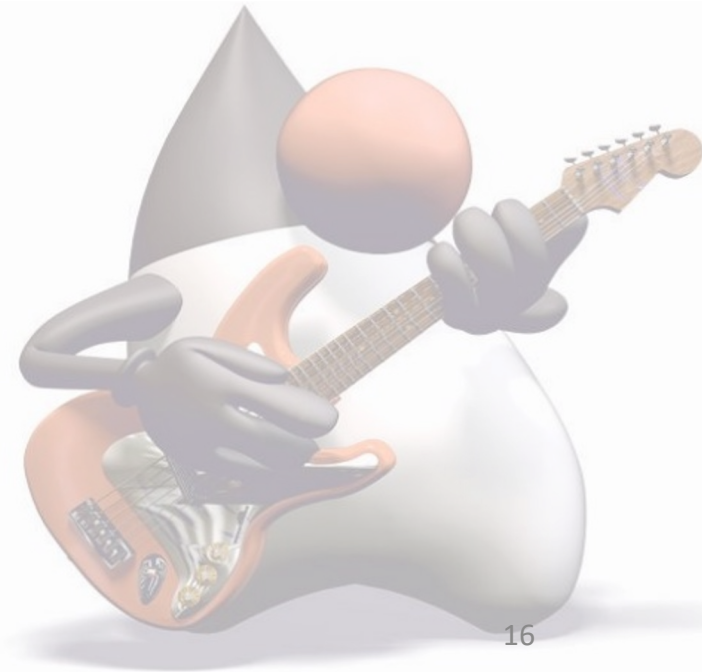


Objekte in Strings konvertieren

Schöner ist es natürlich, die Daten in einem Format auszugeben mit dem der User/Programmierer auch etwas anfangen kann:

```
public class Bank {  
    public String name = "Bank Austria";  
    public String address = "1010 Wien";  
  
    @Override  
    public String toString(){  
        return "Bank: " + name + ", " + address;  
    }  
  
    public static void main(String[] args) {  
        Bank bank = new Bank();  
        System.out.println(bank);  
    }  
}
```

Bank: Bank Austria, 1010 Wien



Übung 5

- Implementieren sie folgende Klassen

- Account
- methods: `withdraw(double amount)`, `deposit(double amount)`, `transfer(Account receiver, double amount)`, `toString()`
 - Constructors: `Account(double initialBalance, Customer, Bank)`
- **Bank**
- **Customer**

- Verhindere das die account balance unter 0 geht
- Schreibe eine Main Methode die verschiedene Accounts erstellt
- Verwenden sie alle Methoden die sie geschrieben haben
- Jede Methode sollte am Ende das Ergebnis der Operation ausgeben (`toString()`)

Konto
iban: String kontoinhaber: String kontostand: long
Konto(iban: String, kontoinhaber: String, initialerKontostand: long) abheben(betrag: long):long einzahlen(betrag: long): long ueberweisen(konto: Konto, betrag: long):long toString():String setInhaber(inhaber:String):void getInhaber():String getIban():String getKontostand():long

