

Beispiel Konto-Klasse

- Die Klasse Konto ist so zu erweitern, dass man
 - ⇒ beliebige Objekte mit vorgegebenen Attributen anlegen kann und dass
 - ⇒ beim Anlegen geprüft wird, ob korrekte Werte übergeben wurden.
 - ⇒ Ebenso sind die Methoden um Prüfungen zu ergänzen.

Konstruktoren

- Klassen benötigen sogenannte **Konstruktoren**, um Objekte anlegen zu können.
 - ⇒ Jede Klasse hat immer mindestens einen **Konstruktor**.
 - ⇒ Ist kein Konstruktor explizit definiert, so wird vom Compiler ein parameterloser Konstruktor, der sogenannte **Standard-Konstruktor** (default constructor) erzeugt.
 - ⇒ Konstruktoren heißen genau wie die Klasse und unterscheiden sich durch Anzahl und Typ der Übergebenen Parameter.

```
public Konto () {  
}
```

- ⇒ Ist ein anderer Konstruktor vorhanden, so wird nicht automatisch ein Standard-Konstruktor angelegt.

Konstruktoren mit Parametern

- Eine Klasse kann theoretisch beliebig viele Konstruktoren besitzen.
- Konstruktor mit Parametern:

```
public Konto(int kontonr, String inhaber,  
             double kontostand) {  
    this.kontonr = kontonr;  
    this.inhaber = inhaber;  
    this.kontostand = kontostand;  
}
```

- ⇒ Damit man Attribut von Parameter unterscheiden kann, muss beim Attribut der Präfix **this** vor den Attributnamen geschrieben werden.
- ⇒ **this** ist intern eine Referenz auf das aktuelle Objekt, in dem man sich befindet.

Konstruktoren gegenseitig aufrufen

- Wir können nun noch einen zweiten Konstruktor definieren, bei dem nur zwei Parameter übergeben werden.

```
public Konto(int kontonr, String inhaber) {  
    this(kontonr, inhaber, 0.0);  
}
```

- ⇒ **this** bedeutet hier, dass als erste Aktion in diesem Konstruktor ein anderer Konstruktor derselben Klasse aufgerufen wird.
- ⇒ Das hat den Vorteil, dass nur ein Konstruktor alle Initialisierungen vornehmen muss und die anderen sich darauf beziehen können.

Objekte anlegen

- Der **new-Operator** ruft automatisch den passenden Konstruktor der Klasse auf, um ein Objekt anlegen zu können.

```
// 1. Konstruktor  
Konto k1 = new Konto(4711, "Meier, Hans", 1000.0);  
  
// 2. Konstruktor  
Konto k2 = new Konto(4712, "Müller, Fritz");
```

Prüfungen ergänzen

- Problem: Was passiert, wenn wir dem Konstruktor falsche Werte übergeben, z. B. eine negative Kontonummer oder ein Leerstring als Namen?
- Frage: Wie können wir verhindern, dass ein Objekt mit falschen Attributwerten angelegt wird?
- Ein ähnliches Problem stellt sich, wenn einer Methode ein falscher Wert übergeben wird.
 - ⇒ Wie reagiert man, wenn der Methode einzahlen() der Wert -1000.0 übergeben wird?
 - ⇒ Macht es Sinn eine Fehlermeldung auf die Standardausgabe auszugeben?

Prüfungen im Konstruktor

```
public Konto(int kontonr, String inhaber,
              double kontostand){
    if (kontonr > 999 && kontonr < 10000) {
        this.kontonr = kontonr;
        this.inhaber = inhaber;
        this.kontostand = kontostand;
    } else
        System.out.println("Kontonummer nicht 4-stellig");
}
```

- Warum ist diese Prüfung unsinnig?
- Wohin geht denn diese Ausgabe?
- Was ist mit dem Objekt, wird es trotzdem angelegt?

Konzept der Zusicherungen

- Die Java Version 1.4 hat das Konzept der **Zusicherungen** (Assertions) als Sprachelement aufgenommen.
- Eine **Zusicherung** ist eine Aussage über das laufende Programm, die der Programmierer für erfüllt hält.
- Ist diese Aussage wider Erwarten nicht erfüllt, wird eine sogenannte **Ausnahme** ausgelöst.

Die assert-Anweisung

Syntax:

```
assert Boolescher-Ausdruck : [Ausdruck];
```

Wirkungsweise:

- Der boolesche Ausdruck wird ausgewertet
- Ist er **true**, so wird mit der nächsten Anweisung fortgefahren
- Ist er **false**, so wird die Ausnahme **AssertionError** ausgelöst. Der Programmablauf wird abgebrochen, es sei denn es gibt eine sogenannte Ausnahmebehandlung (dazu später)

Assertions in Java

- Assertions sind standardmäßig "ausgeschaltet". Damit man sie verwenden kann, muss man sie explizit aktivieren.
- Die Übersetzung muss mit einer expliziten Option erfolgen (ist mittlerweile aber voreingestellt):

```
javac -source 1.4 Konto.java KontoTest.java
```

- Die Ausführung muss mit einer expliziten Option erfolgen:

```
java -ea KontoTest      (oder alternativ)  
java -enableassertions KontoTest
```

Anwendung im Konstruktor

```
public Konto(int kontonr, String inhaber,
             double kontostand) {
    assert kontonr > 999 && kontonr < 10000
        : "Kontonr muss 4-stellig sein";
    this.kontonr = kontonr;
    this.inhaber = inhaber;
    this.kontostand = kontostand;
}
```

- Ablauf bei einer gültigen Parameterübergabe:
 - ⇒ das Programm läuft nach assert normal weiter
- Ablauf bei einer ungültigen Parameterübergabe, z.B. einer 3-stelligen Zahl:
 - ⇒ assert löst eine Ausnahme AssertionError aus mit der angegebenen Fehlermeldung.
 - ⇒ folgende Anweisungen werden übersprungen.
 - ⇒ Das bereits allokierte Objekt wird wieder freigegeben.

Objekte ausgeben

- Wie geht man am Besten vor, wenn ein Objekt ausgegeben werden soll?
- Dazu gibt es mehrere Lösungsmöglichkeiten:
 1. Eine Ausgabemethode in die Klasse Konto einbauen.
 2. Für alle Attribute get-Methoden schreiben und diese bei Ausgaben verwenden.
 3. Einfach mal ausprobieren, ob man das Objekt direkt ausgeben kann.

```
public class KontoTst {
    public static void main(String[] args) {
        Konto k1 = new Konto(4711, "Meier", 2000.0);
        k1.einzahlen(1000.0);
        System.out.println("Konto1: " + k1);
    }
}
```

Die toString-Methode

- Die **toString**- Methode ist eine Methode, die von der Klasse **java.lang.Object** geerbt wird (mehr dazu später).
- Damit besitzt jede Klasse automatisch eine toString-Methode.
- Aufgabe: textuelle Aufbereitung des Objektinhaltes.
- Standardimplementierung: Name der Klasse + Adresse des Objektes im Hauptspeicher.

Konto1: Konto@54520eb

Die toString-Methode

- Redefinition der **toString**-Methode in der Beispielsklasse Konto:

```
public String toString() {  
    return "Kontonr : " + kontonr +  
           ", Inhaber: " + inhaber +  
           ", Kontostand: " + kontostand;  
}
```

- Ausgabe im Testprogramm:

Konto1: Kontonr : 4711, Inhaber: Meier, Kontostand: 3000.0

Methoden

- Es gibt **(Objekt-)Methoden**, also Methoden, die über die `this`-Referenz auf Objektattribute zugreifen können wie z. B. *einzahlen* und *abheben* in der Klasse `Konto`.
 - ⇒ Aufruf
`Objektname.Methodenname (Par1, Par2, ...)`
- Es gibt **Klassen-Methoden**, zu erkennen an dem Zusatz **static**.
 - ⇒ Kein Zugriff auf Objektattribute
 - ⇒ Nur Zugriff auf Klassenattribute (Zusatz `static`).
 - ⇒ Aufruf
`Klassenname.Methodenname (Par1, Par2, ...)`