- Recap, Feinschliff und ein bisschen was Neues -

Überblick

- Einführung in das Programmieren (S. 1-140, S. 181-226, = 185 Seiten)
  - Grundlagen (Kapitel 1, im Schnelldurchlauf!)
  - Datentypen und Operatoren (Kapitel 2)
  - Kontrollanweisungen (Kapitel 3)
  - Arrays (Kapitel 5)
- Objektorientierte Programmierung (S. 141-180, S. 227-388, = 200 Seiten)
  - Klassen, Objekte Methoden (Kapitel 4)
  - Genauere Betrachtung der Methoden und Klassen (Kapitel 6)
  - Vererbung und Polymorphie (Kapitel 7)
  - Pakete und Schnittstellen (Kapitel 8)
  - Ausnahmebehandlung (Kapitel 9)

...nochmal kurz zur Rekursion

- Musterbeispiel Fakultät: n! = n * (n-1)!, 1! = 1
- Rekursion ermöglicht oft elegante Lösungen

```
class Factorial {
    long calculateFactorial(long n) {
        if (n <= 1) {
            return 1;
        } else {
            return (n * calculateFactorial(n-1));
        }
    }
}

class FactorialDemo {
    public static void main(String[] args) {
        Factorial f = new Factorial();
        System.out.println("9! = " + f.calculateFactorial(9));
    }
}
```

...nochmal kurz zu Objektvariablen

- "`null`"
  - Objektvariable mit dem Wert `null` verweist auf *kein* Objekt
  - per `null`-Zuweisung werden referenzierte Objekte abgehängt
  - was, wenn es keine weiteren Referenzen auf ein Objekt gibt?
  - `null` ist nicht gleichbedeutend mit "uninitialisiert"

- Identity vs. Equality
  - der == Operator betrachtet den Variableninhalt
  - bei Objekt-Variablen ist das die *Referenz*, nicht das Objekt!

```
String a = new String("Some String");
String b = new String("Some String");
String c;

System.out.println(a == b); // check for Identity: false!
System.out.println(a.equals(b)); // check for Equality: true
```

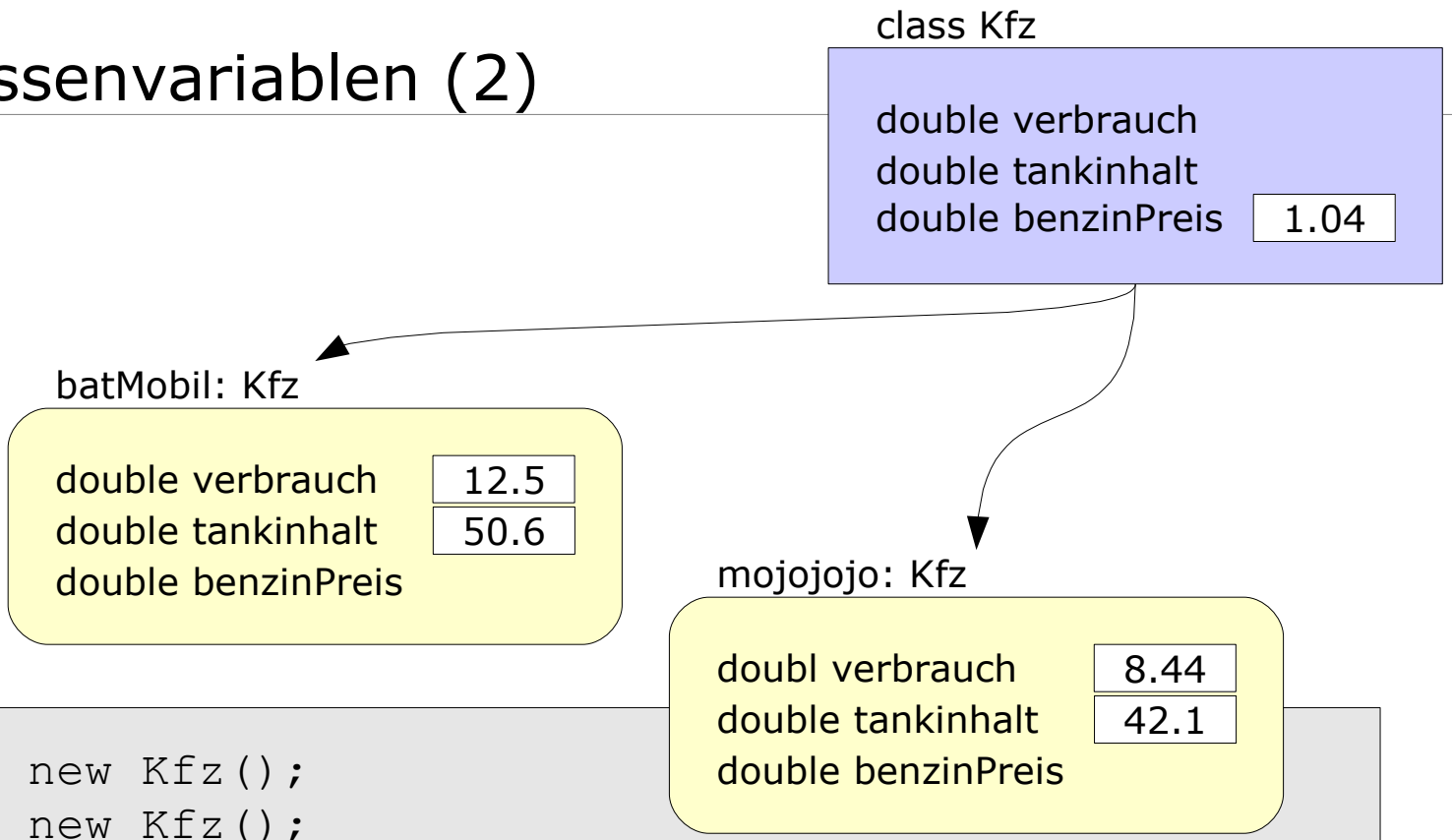
Instanz- vs. Klassenvariablen (1)

- Instanzvariablen existieren in der Instanz
  - alter Hut: jede Instanz bekommt eine frische Kopie der Variable
  - jede Instanz speichert darin ihren eigenen Wert

- Klassenvariablen existieren in der Klasse
  - Deklaration per Schlüsselwort "static"
  - es existiert nur *eine* Variable
  - jede Instanz greift auf *dieselbe* Variable zu
  - wichtig: Zugriff auch ohne Instanz möglich: Klasse.variable;

```
class Kfz {
    int sitzplaetze;
    double verbrauch;
    double tankinhalt;
    static double benzinPreis = 1.04; // für alle Instanzen
}
```

Instanz- vs. Klassenvariablen (2)




```
Kfz batMobil = new Kfz();
Kfz mojojojo = new Kfz();
//...
batMobil.benzinPreis = 0.99;
mojojojo.benzinPreis = 1.04; // dieselbe Variable!

System.out.println(batMobil.benzinPreis); // 1.04
System.out.println(Kfz.benzinPreis); // Zugriff ohne Instanz!
```

Konstanten deklarieren

- weiteres Schlüsselwort "final"
  - nachdem einer final Variable einmal ein Wert zugewiesen wurde, ist der Wert dieser Variable nicht mehr änderbar

- "final" in Verbindung mit "static"
  - Deklaration von leicht zugänglichen Konstanten
  - Konvention: Konstanten in GROSSBUCHSTABEN

```
class Math {
    static final double PI = 3.14159;
    static final double E = 2.71828;
    // ...
}

double umfang = durchmesser * Math.PI;
```

auch Methoden können statisch sein

- statische Methoden...
  - werden direkt in der Klasse aufgerufen: Klasse.methode()
  - können also ohne Instanz einer Klasse ausgeführt werden
  - haben deshalb keinen Zugriff auf Instanzvariablen oder nicht-statische Methoden!

- Frage: wann/wofür machen statische Metoden Sinn?

- prominentes Beispiel: main()
  - wenn der Interpreter beim Start eines Programms diese Methode aufruft, gibt es noch keine Instanz der Klasse!

```
static final double d = 1.234;
public static void main(String[] args) {
    System.out.println(Math.pow(d, Math.PI));
}
```

Access Modifiers, Information Hiding (1)

- Attribute, Konstruktoren und Methoden kontrollieren per Access Modifier, wer auf sie zugreifen/sie aufrufen darf
  - private: Zugriff nur von der eigenen Klasse aus
  - public: Zugriff auch von anderen Klassen aus

- Accessors und Mutators
  - oft private Instanzvariablen mit öffentl. Methoden für Zugriff
  - Vorteil: bessere Kontrolle über den eigenen Zustand
  - nennt man wegen ihrer Namen auch: "Getter" und "Setter" (siehe nächste Folie)

- Information Hiding
  - eine Klasse *braucht* Interna einer anderen Klasse nicht kennen
  - eine Klasse *darf* Interna einer anderen Klasse nicht kennen
  - somit keine Abhängigkeiten von speziellen Implementierungen
  - lose Kopplung: it's a Good Thing, u.a. für die Wartung

Access Modifiers, Information Hiding (2)

- Beispiel

```
class Person {

    private int age;

    // Accessor Methode
    public int getAge() {
        return age;
    }

    // Mutator Methode: keine negativen Werte für "age"
    public void setAge(int newAge) {
        if (newAge >= 0) {
            age = newAge;
        }
    }
}
```

Schnittstellen und Dokumentation (1)

- nützliche Klassen werden oft in verschiedenen Kontexten bzw. Projekten wiederverwendet
  - von Programmierern, die diese Klassen nicht geschrieben haben
  - wichtig: Schnittstelle ("wie arbeite/spreche ich mit der Klasse?")
  - zur Schnittstelle gehören *alle öffentlichen Elemente* einer Klasse

- Dokumentation
  - gute Dokumentation ist ein Qualitätsmerkmal
  - gut bedeutet: leicht lesbar
  - leicht lesbar bedeutet:
    - standardisiertes Format
    - strukturiert
    - beschreiben, WAS passiert
    - nicht beschreiben, WIE es passiert

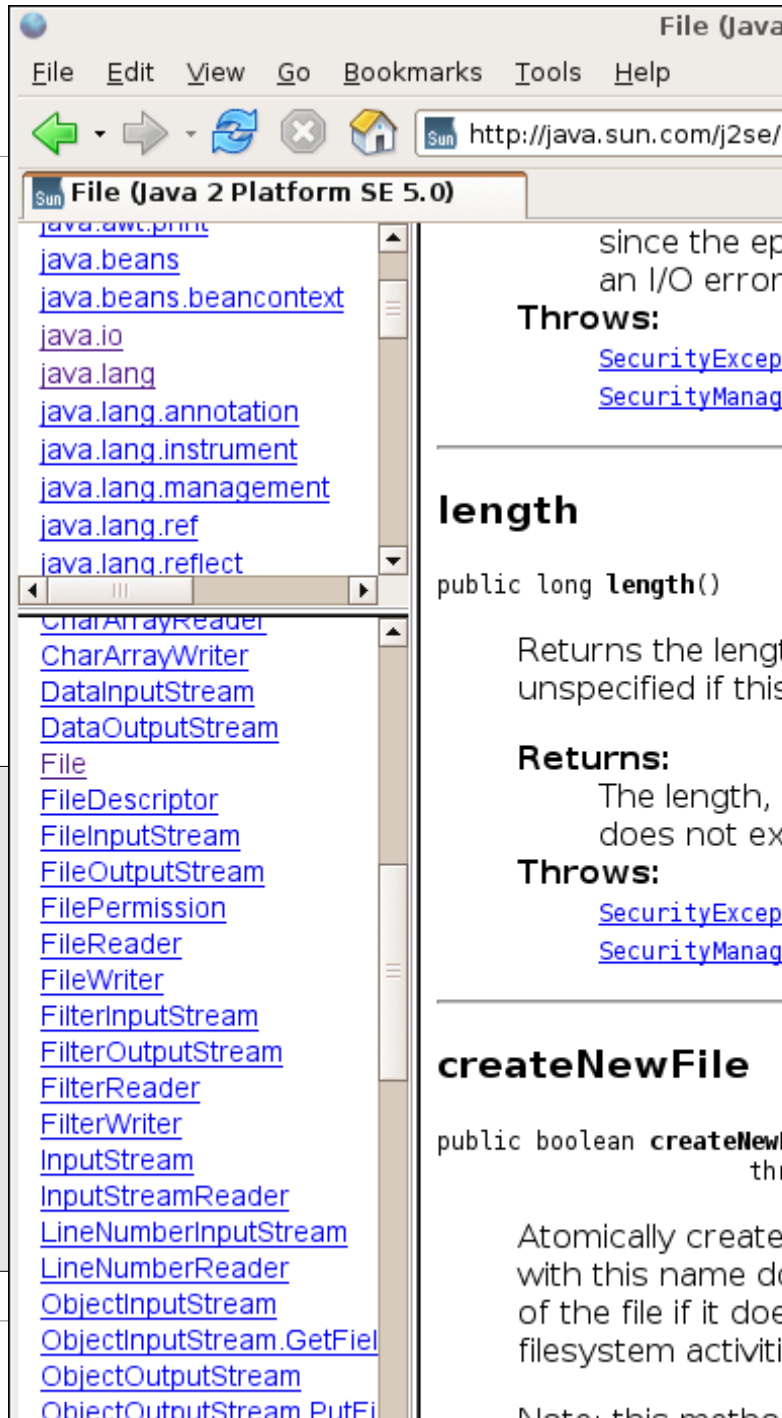
Schnittstellen und Dokumentation (2)

- Dokumentation einer Klasse wird direkt aus dem Quelltext erzeugt:
  - aus den Signaturen, und
  - aus speziell markierten Kommentaren:
    - /** ... */
    - @param, @return, @author, ...

- Werkzeug dafür: javadoc.exe

```
public class File {
     // ...

     /**
       * Returns the length of the file denoted by...
       * @return The length, in bytes, of the file...
       */
     public long length() {
          // ...
     }
}
```



näheres zur Java Klassenbibliothek und zu Packages (1)

- Java Klassenbibliothek ist Teil der JRE
  - Klassen der KB stehen somit *jedem* Programm zur Verfügung
  - Klassen der KB sind thematisch in Packages gruppiert
  - Struktur-Analogie: Package/Klasse vs. Verzeichnis/Datei
  - im JDK sind zusätzlich die Quelltexte aller Klassen enthalten

- Klassen aus anderen Packages im Quelltext verwenden
  - entweder vollständige Angabe des Klassennamen samt Package,
  - oder Import von Klassen mittels import-Anweisung
    - ermöglicht kürzere Schreibweise im Quelltext
    - Kurzschreibweise beim Import: import java.util.*;
  - Klassen aus Package java.lang sind immer bekannt (!)

näheres zur Klassenbibliothek und zu Packages (2)

- Klassen ohne Angabe einer Package-Zugehörigkeit (wie alle unsere Klassen bisher) befinden sich im selben, sogenannten 'namenlosen Package'

- Klassen sollten immer gepackaged werden
  - dazu: package-Anweisung am Anfang der Quelltext-Datei
  - Package-Namen sollten eindeutig sein (theoretisch global!)
  - Konvention zur Vergabe von Package-Namen:
    - zuerst 'rückwärts geschriebene URLs' für grobe Zuordnung,
    - dann tiefere Hierarchien für Projekte, Komponenten, ...

- Beispiele für Package-Namen:
  - org.apache.commons.lang
  - com.ait.punchline.frontend
  - org.gudy.azureus2.ui
  - ...

*.jar-Dateien: Java Archives (Jars)

- funktionieren wie Zips
  - Jars lassen sich z.B. per WinZip öffnen
  - besser: jar-Tool zum Erzeugen/Bearbeiten von Jars im SDK
- gruppieren *.class-Dateien, die zu einem Programm, einer Bibliothek, einer Komponente, etc. gehören
  - auch die Java KB kommt als Jar: ...\jre\lib\rt.jar
  - z.B. werden KBs von Drittanbietern als Jars ausgeliefert
  - siehe auch: jedit.jar, jalopy.jar, etc.
- können in den Classpath aufgenommen werden
  - Jars sind wie Verzeichnisse Container für *.class Dateien
  - set CLASSPATH=commons-lang.jar;h:\java;

Beispiel für Verwendung von externen Klassen

- externe Klassen v. http://jakarta.apache.org/commons/lang

```
import org.apache.commons.lang.StringUtils;
class Center {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println(StringUtils.center(args[i],20));
        }
    }
}
```

```
c:\java> set CLASSPATH=commons-lang-2.1.jar;c:\java
c:\java> javac Center.java
c:\java> java Center a bbb ccccc
          a
         bbb
        ccccc
```