Fachbereich Automatisierung und Informatik	HOCHSCHULE harz Hochschule für angewandte Wissenschaften (FH) Wernigerode

„Java“

Grundlagen

Nachschlage-Script

Dipl. Inf., Dipl.-Ing. (FH) Michael Wilhelm
Friedrichstraße 57 - 59
38855 Wernigerode

Raum: **2.202**
Tel.: **03943/659-338**
Fax: **03943/659-399**
EMail: **mwilhelm@hs-harz.de**

Inhaltsverzeichnis

1 Grundlagen

1.1 Datentypen

1.1.1 Ganzahlig mit Vorzeichen:

- byte 1 Byte Länge -128 bis 127
- short 2 Byte Länge -32768 bis 32767
- int 4 Byte Länge 2.147.483.648 bis 2.147.483.647
- long 8 Byte Länge 9.223.372.036.854.775.808 9.223.372.036.854.775.807

1.1.2 Nachkommazahlen

- float 4 Byte Länge
- double 8 Byte Länge

1.1.3 Wahrheitswerte

- boolean 1 Byte Länge

1.1.4 Zeichenfolgen

- char 2 Byte
- String, Klasse 2 Byte pro Zeichen

1.1.5 Methoden der Klasse String

Allgemeine Methoden:

- char charAt(int index);
- String substring(int begin, int end);
- String trim();
- int length();

Vergleichen von Zeichenketten:

- boolean equals(Object anObject);
- boolean startsWith(String s);
- boolean endsWith(String s);
- int compareTo(String s);

Suchen in Zeichenketten:

- int indexOf(String s);
- int indexOf(String s, int fromIndex);

- int lastIndexOf(String s);

Ersetzen von Zeichenketten:

- String toLowerCase();
- String toUpperCase();
- String replace(char oldchar, char newchar);

Konvertierungsfunktionen (Datentyp nach String):

- static String valueOf(boolean b);
- static String valueOf(char c);
- static String valueOf(char c[]);
- static String valueOf(char[] data, int offset, int count)
- static String valueOf(double d);
- static String valueOf(float f);
- static String valueOf(int i);
- static String valueOf(long l);
- static String valueOf(Object obj);

Konvertierungsfunktionen (String nach Datentyp):

```
String sValue="123.44";
int n;
try {
  n= Integer.parseInt(sValue);
  // double f = Double.parseDouble(sValue);

}
catch ( NumberFormatException e ){
  System.err.printf( "Konvertierungsfehler: '%s' ",sValue );
}
```

Alternativ:

```
String sValue="123.44";
int n;
try {
  n = Integer.valueOf(sValue).intValue();
}
catch (NumberFormatException e) {
  System.err.printf( "Konvertierungsfehler: '%s' ",sValue );
}
```

Weitere Methoden:

- parseBoolean(String s)
- parseByte(String s)
- parseShort(String s)
- parseInt(String s)
- parseLong(String s)
- parseDouble(String s)
- parseFloat(String s)

1.1.6 Wichtige Escape-Sequenzen

Zeichen	Bedeutung
\n	Zeilenumbruch Java, Apple, Unix
\r\n	Zeilenumbruch Windows
\f	Seitenumbruch (Formfeed)
\r	Wagenrücklauf (Carriage return)
\t	Tabulator
\"	Doppeltes Anführungszeichen
\'	Einfaches Anführungszeichen
\\	Backslash

1.2 Arrays

Array sind Datenstrukturen, die mehrere Elemente **eines** primitiven Datentyps speichern können.

Deklaration:
Datentyp[] variablenname;

Erzeugen des Arrays:
Variablenname = new Datentyp[anzahl];

Hinweis:
- Der Index läuft immer von Null bis n-1.
- Bei Objekten muss man aber erst die Elemente erzeugen.

Beispiel:
```
int[]  feld;
feld = new int[3];
feld[0] = 1;
feld[1] = 3;
feld[2] = 2;

Ausgabe mittels for-Schleife:
for (int i=0;  i< feld.length; i++) {
   System.out.println( "feld["+i+"] = " + feld[i]);
}
Ausgabe mittels for-each-Schleife:
for (int k : feld) {
   System.out.println( "feld[] = " + k);
}
```

1.2.1 Arrays und Methoden

Arrays sind Objekte, so dass sie immer als Referenz übergeben werden.

Beispiel:
```
int[]  feld;
feld = new int[3];
feld[0] = 1;
feld[1] = 3;
feld[2] = 2;
float xm = mittelwert(feld);

private float mittelwert( int [] feld )  {
  float summe=0.0f;
```

```
  float xm=0.0f;
  for (int k : feld) {
    summe+=k;
  }
  xm = summe/feld.length;
  return xm;
}
```

1.3 Abfragen

`if (Bedingung)`
`  Anweisung;`
	`if (Bedingung) {`
	`  Anweisung;`
	`}`
		`if (Bedingung) {`
		`  Anweisung1;`
		`  Anweisung2;`
		`}`

`if (Bedingung)`
`  Anweisung1;`
`else`
`  Anweisung2;`
	`if (Bedingung) {`
	`  Anweisung1;`
	`}`
	`else {`
	`  Anweisung2;`
	`}`
		`if (Bedingung) {`
		`  Anweisung1;`
		`  Anweisung2;`
		`}`
		`else {`
		`  Anweisung3;`
		`  Anweisung4;`
		`}`

Beispiele:

`if (i<10) {`
`  a=33;`
`}`
	`if (i<10) {`
	`  a=33;`
	`}`
	`else {`
	`  a=12;`
	`}`
		`if (i<10) {`
		`  a=33;`
		`  b=12;`
		`}`
		`else {`
		`  a=12;`
		`  b=33;`
		`}`

`if (i<0) {`
`  a=-1;`
`}`
`else {`
`  if (i>0) {`
`  a=+1;`
`  }`
`  else {`
`  a=0;`
`  }`
`}`
	`if (i < 0) {`
	`  a = -1;`
	`} else if (i > 0) {`
	`  a = +1;`
	`  } else {`
	`  a = 0;`
	`  }`
		`if (i < -10) {`
		`  a = -2;`
		`} else if (i < 0) {`
		`  a = -1;`
		`} else if (a==0){`
		`  a = 0;`
		`} else if (a<10){`
		`  a = 1;`
		`} else {`
		`  a=2;`
		`}`

1.4 Schleifen

1.4.1 While-Schleife

Init der Bedingung
while (Bedingung) {
  Aktion
  Increment oder Decrement etc.
}

Beispiel:
```
int i=10;
while (i>0) {
   System.out.println("i: "+i);
   i--;
}
```

1.4.2 For-Schleife

for (Anfangsinit ; Bedingung ; Increment oder Decrement etc.) {
   Aktion
}

Beispiele:

```
for (int i=0;  i<10; i++) {
   System.out.println("i: "+i);
}

for (int i=10;  i>0; i--) {
   System.out.println("i: "+i);
}
```

Eine For-Schleife ist immer sinnvoll bei festen Grenzen. Eine While-Schleife benutzt man, wenn man die Schleifendurchläufe nicht berechnen kann. Also eine Abbruchbedingung:

```
int  x;
x = read();
while ( x > 0) {
  // Aktion
  x = read();
}
```

1.4.3 For-each-Schleife

for (Datentyp variable : Collection {
   Aktion mit der Variablen „variable“
}

Beispiel:
```
int[]  feld;
feld = new int[3];
feld[0] = 1;
feld[1] = 3;
feld[2] = 2;

for (int k  :  feld ){
   System.out.println( "feld[] = " + k );   // ein Index gibt es nicht mehr
}
```

Hinweis:
- Benutzt auch bei ArrayList, Vector, LinkedList und Stack

2 Rekursion

Grundlegende Schritte der Rekursion:

- Reduktion auf kleinerere Probleme.
- Lösung für kleinste Probleme bekannt.

2.1 Beispiele:

rekursiv definierte Funktionen:
a) Fakulatät n! = n * (n-1)! falls n > 1,
1! = 0! = 1

Java-Code:

```
long calcFakulteat(int n) {
   long result = 1;
   if(n > 1) {
     result = n * calcFakulteat(n-1);
   }
   else if (n < 0) {
     result = 0;
   }
   return result;
}
```

Benutzung
Anzahl der Möglichkeiten n unterscheidbare Elemente anzuordnen
b) Fibonacci-Folge: f(n) = f(n-1) + f(n-2) falls n > 1,
f(1) = 1,
f(0) = 0
Benutzung
Wachstum (Wachstum einer Kaninchenpopulation)
Abstand der Blätter

Java-Code:

```
long calcFibonacci2(int n) {
   long result = 0;
   if(n > 1) {  // Verhindert eine unendliche Rekursion
     result = calcFibonacci2(n-1)+calcFibonacci2(n-2);
   } else if(n == 1) {
     result = 1;
   }
   return result;
}
```

3 Klassen, Vererbung und Schnittstellen

3.1 Klasse

Eine Klasse kapselt Attribute und Methoden

3.1.1 Deklaration

```
class Klassenname  {
    // Attribute
    // Default-Konstruktor wird automatisch erzeugt, wenn kein anderer definiert wurde
}
```

3.1.2 Attribute

Attribute können folgende Eigenschaften besitzen:

- private Zugriff nur in der Klasse
- protected Zugriff nur in der Klasse und abgeleiteten Klassen
- public Allgemeiner Zugriff (verboten)

Der Zugriff sollte nur über set-und-get-Methoden erfolgen:

- Methoden zum Lesen und Schreiben von Attributen heißen getter/setter Methoden. Ihre Namen folgen den Konventionen für Funktionen, wobei der erste Namensteil entweder set oder get ist, der zweite Namensteil ist der Name des Attributs.
- In Eclipse kann man das automatisch generieren lassen
  - Das Attribut auf „private“ setzen.
  - Mit der Maus über das Attribut gehen.
  - Eintrag „Create getter and setter for 'alter' “ auswählen.

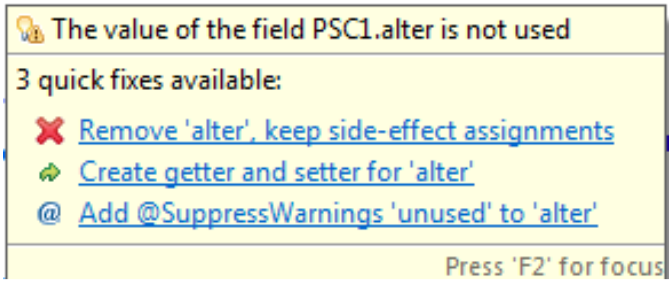


Abbildung 1 Eclipse: set-getter

Netbeans:

- Taste Alt+Einfg

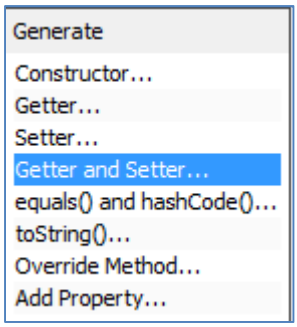


Abbildung 2 Netbeans set-getter Aufruf

Abbildung 3 Netbeans set-getter Dialog

3.1.3 Konstruktor

Konstruktoren können andere Konstruktor in der eigenen Klassen aufrufen.
Vorteil:

- Eine Zuweisung
- Keine Code-Redundanz

Beispiel:

```
class Student {
   int mnr;
   String name;
  // default Constructor
  public Student(){
   this(0,"noname");
  }

  // 2. Konstruktor
  public Student(int mnr, String name){
   this.mnr = mnr;
   this.name = name;
  }
  // Copy-Konstruktor
  public Student (Student std){        // Copy-Konstruktor
   this.mnr = std.mnr;
   this.name = std.name;
  }
}
```

3.2 Vererbung

- Klassen können von anderen Klassen abgeleitet werden
- **Sie „erben“ alle Attribute**
- **Sie „erben“ alle Methoden**
- Methoden können aber „abgelehnt“ werden
- Man kann „Sub“-Klassen zwingen, Methoden zu implementieren
- Es entsteht keine Code-Redundanz
- Das Schlüsselwort lautet **extends**
- In Java ist nur die Einfach-Vererbung möglich.
    - **Ein** Parent.

```
class Class1 {
   private String name;

   public Class1 (String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name=name;
   }
}

class Class2 extends Class1 {
   private int nummer;

   public Class2 (String name, int nummer) {
      super(name);    // muss immer als erstes aufgerufen werden
      this.nummer = nummer;
   }

   public int getNummer() {
      return nummer;
   }
   public void setNummer(int nummer) {
      this. nummer =nummer;
   }
   // die Attribute und Methoden der Oberklassen werden vererbt
}
```

3.3 Abstrakte Klassen

- Um Sub-Klassen zu zwingen, Methoden zu implementieren, kann man eine sogenannte abstrakte Klasse oder ein Interface (Siehe Kapitel 3.5,Seite 14) definieren
- Eine abstrakte Klasse kann nicht instanziiert werden
- Eine von einer abstrakte Klasse abgeleiteten Klasse **kann** instanziiert werden
- Eine abstrakte Klasse kann auch als Datentyp in einem Array dienen
- Eine abstrakte Klasse kann auch Attribute haben, der Hauptzweck sind aber die Methoden-definitionen

```
abstract class GeoObject {
   abstract float getFlaeche();
}

class Quadrat extends GeoObject {
   private float a;
   public Quadrat  (float a) {
      this.a = a;
   }

   public float getFlaeche () {
      return a*a;
   }
}  // Quadrat

class Rechteck extends GeoObject {
   private float a;          // Eine Variable, eine Zeile
   private float b;
   public Rechteck  (float a, float b) {
      this.a = a;
      this.b = b;
   }

   public float getFlaeche () {
      return a*b;
   }
}  // Rechteck
```

3.4 Polymorphismus, Vielgestaltigkeit

- Abstrakte Methoden müssen immer implementiert werden.
- Dynamische Methoden können implementiert werden.
- Java sucht im aktuellen Objekt, zum Beispiel „Quadrat“ nach der gesuchten Methode.
- Wenn Java sie findet, okay
- Falls nicht, sucht Java sie in die nächsthöhere Klasse, bis eine Methode gefunden wurde.
- Die Polymorphie macht es möglich, dass verschiedene Sub-Klassen dieselbe Methode verstehen, obwohl der technische Aufruf auf diese Anfrage, z. B. draw, völlig unterschiedlich sein kann.
- Auf die Botschaft „draw“ können sowohl Objekte vom Typ „Auto“, als auch vom Typ „Fahrrad“ reagieren.
- In jedem Fall wird **eine** Methode aufgerufen.
- Welche entscheidet Java zur Laufzeit.
- Polymorphie wird nun dadurch realisiert, dass ein Objekt eine geerbte Methode, draw, abändern kann, um in der gewünschten Weise zu reagieren. Diesen Vorgang nennt man auch **Überschreiben** einer Methode.
- Polymorphismus funktioniert mit abstrakten und dynamischen Klassen.
- Überschreiben verwendet mehrere Klassen.

```
abstract class GeoObject {
   abstract float getFlaeche();
}

class Quadrat extends GeoObject {
   private float a;
   public Quadrat  (float a) {
      this.a = a;
   }

   public float getFlaeche () {
      return a*a;
   }
```

```
}  // Quadrat

class Rechteck extends GeoObject {
   private float a;          // Eine Variable, eine Zeile
   private float b;
   public Rechteck  (float a, float b) {
      this.a = a;
      this.b = b;
   }

   public float getFlaeche () {
      return a*b;
   }
}  // Rechteck


private void test() {
   // Array mit 5 Geoobjekten
   // nur ein Platz für 5 GeoObjecte, keine Objekte
   GeoObject[] feld = new GeoObject[5];
feld[0] = new Quadrat(3);
feld[1] = new Rechteck(1,4);
feld[2] = new Quadrat(6);
feld[3] = new Rechteck(2,10);
feld[4] = new Quadrat(10);
// Ausgabe mit einer for-each-Schleife
for (GeoObject gobj : feld)  {
System.out.println("Flaeche: "+gobj.getFlaeche() );
}
}
```

Ausgabe:

- Flaeche: 9.0
- Flaeche: 4.0
- Flaeche: 36.0
- Flaeche: 20.0

3.5 Interface und Schnittstelle

Wie kann sichergestellt werden, dass bestimmte Methoden in einer Klasse implementiert sind?
Java bietet den interface-Konstrukt:

- Eine Schnittstelle (engl. Interface) hat einen Namen (Namensgebung wie für Klassen).
- Jedoch enthält ein Interface keine Methodenimplementationen oder Attribute, sondern lediglich die Prototypen der zu implementierenden Methoden.

Das heißt, ein Interface stellt einen Vertrag über in einer Klasse zu implementierende Methoden dar.
Von einem Interface können **keine Instanzen** erzeugt werden, **aber sehr wohl Referenzen**.
Insbesondere Methodenparameter können Interface-Referenzen sein (callBack-Funktionen).

Unterschied abstrakte Klasse versus Interface

```
class abstract Rechner {
  public int takt;
  public abstract void rechne();
}
interface IRechner {
   public void rechne();
}
```

Beispiel:

```
class Notebook extends Rechner {
  public void rechne() {
    // do some stuff
  }
}

class Desktop implements IRechner {
 public void rechne() {
    // do some stuff
  }
}
```

3.5.1 Interface vs. Vererbung

- Eine Klasse kann nur von einer Oberklasse abgeleitet werden.
- Eine Klasse kann beliebig viele Sub-Klassen haben.
- Eine abstrakte Klasse und ein Interface beschreiben die gewünschte Funktionalität für die Sub-Klassen.
- Eine abstrakte Klasse kann Attribute haben.
- Ein Interface kann Attribute haben, diese sind aber final und static, also Konstanten.
- Die ersten konkreten Sub-Klassen müssen die Methoden implementieren, abstrakte Klassen können sie implementieren, müssen es aber nicht.
- Die weiteren Sub-Klassen dürfen die Methoden überschreiben.
- Eine abstrakte Klasse kann nicht erzeugt werden.
- Eine Schnittstelle kann nicht erzeugt werden.
- Eine Klasse kann beliebig viele Interfaces implementieren.
- Jede Schnittstelle definiert eine neue Sicht, eine Art Rolle.
- Implementiert eine Klasse diverse Schnittstellen, können ihre Exemplare in verschiedenen Rollen auftreten.
- Schnittstellen können keine Methoden haben, die Quellcode beinhalten.
- Eine abstrakte Klasse kann eine normale Methode hinzufügen.

4 Sortierung

Sollen Instanzen sortiert werden, so muss die Methode „int compareTo“ in der Klasse implementiert werden.

Mathematik	Java	Normaler Wert
a < b	a.compareTo(b) < 0	-1
a = b	a.compareTo(b) = = 0	0
a > b	a.compareTo(b) > 0	+1

Sortieren und Suchen mit Arrays:

- `Arrays.sort(feld);  // sortieren durch Schnittstelle`
- `Student std = new Student(2345, "Schulze");`
- `int index = Arrays.binarySearch(feld, std);`

Sortieren und Suchen mit ArrayList:

- `ArrayList<Student>  liste = new ArrayList<Student>();`
- `Einfügen von Elementen`
- `Collections.sort( liste );  // Automatisches sortieren`
- `int index = Collections.binarySearch(liste, std);`

4.1 CompareTo

Wenn „compareTo“ implementiert wurde, kann man das Array bzw. die Listen sortieren.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

        private void test1() {
           Student[]  feld = new Student[4];
           feld[0] = new Student(3234, "Meier");
           feld[1] = new Student(4234, "Anhalt");
           feld[2] = new Student(2345, "Schulze");
           feld[3] = new Student(1234, "Humbug");
           List lst = Arrays.asList(feld);  // Umwandeln in eine Liste !!!
           Collections.sort( lst );  // Automatisches sortieren
          for (Student std : feld) {
            System.out.println( std );
          }

        }  // test1

        private void test2() {
           ArrayList<Student>  liste = new ArrayList<Student>();
           liste.add( new Student(3234, "Meier") );
           liste.add( new Student(4234, "Anhalt") );
           liste.add( new Student(2345, "Schulze") );
           liste.add( new Student(1234, "Humbug") );
           Collections.sort( liste );  // Automatisches sortieren
           for (Student std : liste) {
             System.out.println( std );
           }

        }  // test2
```

```
class Student   implements Comparable<Student>  {
  private int mnr;
  private String name;

  public int compareTo (Student std) {
     int ret = Integer.compare(mnr, std.mnr);
     if (ret==0) {
        ret = name.compareTo(std.getName());
     }
    return ret;
  }

}
```

4.2 bubbleSort

```
public void bubbleSort(int[] array) {
    boolean swapped = true;
    int n = array.length;
    while(swapped) {
        swapped = false;
        n--;
        for(int i=0; i<n; i++) {
            if(array[i] > array[i+1]) {
                int temp = array[i];
                array[i] = array[i+1];
                array[i+1] = temp;
                swapped = true;
            }  // if
        } // for
    } // while
}
```

5 Suchen

Das Suchen erfolgt entweder über eine Hash-Table oder über das Binary-Search.
Abfolge:

- Sortieren mittels Collection.sort(…)
- Aufruf der BinarySearch-Funktion

5.1 binarySearch

```
int binarySearch ( int[] array, int key ) {
   int u = 0; // untere Grenze
   int o = array.length -1; // obere Grenze
   while( u <= o ) {
      int m = (u + o ) / 2; // lese den Wert des mittleren Index
      if ( array[m] == key ) { // bester Fall, mit CompareTo
         return m;
      } else if ( array[m] > key ) {
         o = m-1; // Wert in linker Haelfte oder nicht vorhanden
      } else {
         u = m+1; // Wert in rechter Haelfte oder nicht vorhanden
      } // if
   } // while
   return -1; // Wert nicht gefunden
}
```

Beispiel:

```
private void test1() {
   Student[]  feld = new Student[4];
   feld[0] = new Student(3234, "Meier");
   feld[1] = new Student(4234, "Anhalt");
   feld[2] = new Student(2345, "Schulze");
   feld[3] = new Student(1234, "Humbug");
   List lst = Arrays.asList(feld);
   Collections.sort( lst );
   Student std = new Student(2345, "Schulze");
   int index = binarySearch (feld, std);
   if (index>=0) {
      System.out.println( "Gefunden: "+feld[index] );
   }
   else {
      System.out.println( "nicht Gefunden: "+std );
   }
} // test1


private int binarySearch ( Student[] array, Student key ) {
   int u = 0; // untere Grenze
   int o = array.length -1; // obere Grenze
   while( u <= o ) {
      int m = (u + o ) / 2; // lese den Wert des mittleren Index
      if ( array[m].compareTo(key) == 0 ) { // bester Fall, mit CompareTo
         return m;
      } else if ( array[m].compareTo(key)  > 0 ) {
         o = m-1; // Wert in linker Haelfte oder nicht vorhanden
      } else {
         u = m+1; // Wert in rechter Haelfte oder nicht vorhanden
      } // if
   } // while
   return -1; // Wert nicht gefunden
}
```

```
// verwendet die Java-Klassen und Methoden
private void test2() {
           Student[]  feld = new Student[4];
           feld[0] = new Student(3234, "Meier");
           feld[1] = new Student(4234, "Anhalt");
           feld[2] = new Student(2345, "Schulze");
           feld[3] = new Student(1234, "Humbug");
           Arrays.sort(feld);   // sortieren durch Schnittstelle
          Student std = new Student(2345, "Schulze");
          int index = Arrays.binarySearch(feld, std);
          if (index>=0) {
              System.out.println( "Gefunden: "+feld[index] );
          }
          else {
              System.out.println( "nicht Gefunden: "+std );
          }

        } // test2
```

5.2 HashTable, HashMap

Dient zum schnellen Suchen eines Elementes. Es ist nicht sinnvoll bei Änderungen!

Definition Hashing:

- Die Objekte werden in einem Feld mit Indizies 0 bis N-1 gespeichert.
- Die einzelnen Speicherpositionen werden Buckets genannt.
- Eine Hashfunktion h : e --> N bestimmt für ein Schlüsselelement e einen Bucket h(e) im Feld, in dem ein mit dem Schlüssel e assoziertes Element gespeichert werden soll.
- Die Wahl der Funktion h hat entscheidenden Einfluss auf die Qualität des Verfahrens.
- Ein Überläufer ist ein Element, welches eine bereits gefüllte Position in der Hashtabelle zum Überlaufen bringt. Bei der Verkettung der Überläufer werden diese für jeden Bucket in einem Array oder einer verketteten Liste untergebracht.
- HashMap in Java: Schlüssel (hier String) und Wert (hier Klasse) getrennt.
- Hashtable ist synchronisiert, HashMap ist das nicht, also schneller.

```
private void test3() {
   ArrayList<Student>  feld = new ArrayList<Student>();
   feld.add( new Student(3234, "Meier") );
   feld.add( new Student(4234, "Anhalt") );
   feld.add( new Student(2345, "Schulze") );
   feld.add( new Student(1234, "Humbug") );
   HashMap array = new HashMap(10000, 0.70f);
   for (Student std : feld) {
      array.put(std.getMatrnr(),std);
   }
   Student std = (Student) array.get( 2345 );
   if (std!=null) {
      System.out.println( "Gefunden: "+std );
   }
   else {
      System.out.println( "nicht Gefunden: "+std );
   }
} // test3
```

6 Generische Klassen

- Klassen können mit einem oder mehreren zusätzlichen Parametern versehen werden. Diese stehen während der Implementierung als Platzhalter für Klassennamen.
- Bei der **Instanziierung** sind dann für diese Platzhalter konkrete Klassen einzusetzen.
- **Generische Klassen dienen der Typsicherheit.**
- **Der Compiler ersetzt beim Übersetzen alle parametrisierten Klassen mit der Klasse Object!**

6.1 Felder und Listen mit Objekten

Bei der Benutzung von Feldern und Listen sollten in Java bevorzugt generische Klassen benutzt werden. Generische Klassen führen den aktuell benutzten Klassentyp in spitzen Klammern auf. Als aktuelle Klassen sind keine atomaren Typen erlaubt - hier muss boxing benutzt werden.
Man spricht von parametrischer Polymorphie:

- ArrayList<T> liste = new ArrayList<T>();
- List<T> liste = new List<T>();

Beispiel:

```
public class GenKlasse<T> {
   private T[] array;
   public GenKlasse(int n) {
      array = (T[]) new Object[n]; // nicht new T[n] !!!
   }
......
}
```

Benutzung:

```
GenKlasse<String> s = new GenKlasse<String>(10);
```

7 Swing

7.1 Elemente

7.1.1 GUI-Elemente in Java (aktive Elemente)

- JLabel Anzeigefeld
- JTextField einzeiliges Editorfeld
- JSpinner einzeiliges Editorfeld mit Drehfeld
- JButton Schalter (Text, Bild)
- JToggleButton Schalter (Text, Bild)
- JRadioButton Element zur Auswahl, korrespondiert mit anderen
- JCheckBox Element zur Auswahl
- ButtonGroup Umfasst Checkbox, Radiobuttons
- JComboBox Aufklappbare Liste
- JList Liste
- JTextArea mehrzeiliger Editor, ASCII
- JTextPane mehrzeiliger Editor, RTF, HTM
- JEditorPane mehrzeiliger Editor, RTF, HTM
- JTable Tabelle
- JScrollBar Scrollbalken
- JTree Anzeige der Elemente in einem Baum

7.1.2 GUI-Elemente in Java (Container)

- JPanel Rahmen
- JTabbedPane Register
- JScrollPane Speichert GUI-Element, Scroll-Mechanismus JSplitPane Automatischer Aufteiler
- Box Vertikal, Horizontal (Layout)

7.1.3 GUI-Elemente in Java (Menüs, Schalter)

- JMenuBar Speichert einzelne Menüs
- JPopupMenu Lokales Menü
- JToolBar Schalterleiste
- JOptionPane Für einfache Dialogboxen (Klasse Dialog)
- JColorChooser Farbe
- JFileChooser Auswahl einer Datei

7.2 JFrame

```
import javax.swing.*;
public class Frame1 extends JFrame {

  //Frame konstruieren
  public Frame1() {
    this.setSize(400, 300);
    this.setTitle("Frame1");
  }

  public static void main(String[] args) {
    Frame1 frame = new Frame1();
    frame.setVisible(true);
  }
}
```

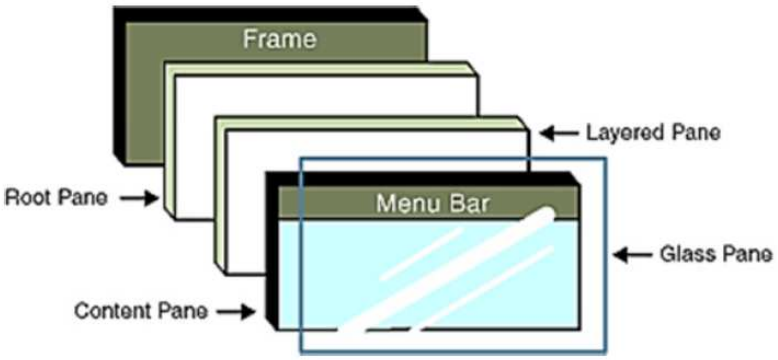


Abbildung 4 In ContentPane werden die Elemente eingetragen

7.2.1 Methode setDefaultCloseOperation

DO_NOTHING_ON_CLOSE

- do not do anything - require the program to handle the operation in the windowClosing method of a registered WindowListener object.

HIDE_ON_CLOSE

- automatically hide the frame after invoking any registered WindowListener objects

DISPOSE_ON_CLOSE

- automatically hide and dispose the frame after invoking any registered WindowListener objects

EXIT_ON_CLOSE

- The exit application default window close operation. If a window has this set as the close operation and is closed in an applet, SecurityException may be thrown. It is recommended you only use this in an application.
- Wichtig:
  - Diese Anweisung darf nur im Hauptfenster eingestezt werden.
  - In Dialo9gen benutzt man HIDE_ON_CLOSE

7.3 Wichtige Swing-Elemente

7.3.1 JLabel

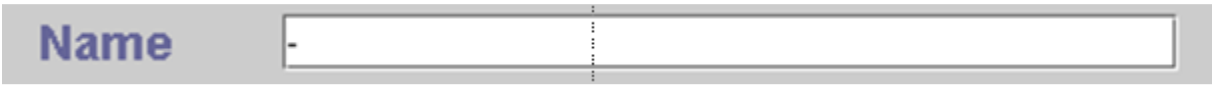
Ein JLabel dient zur Beschriftung von GUI-Elementen.
Es kann eine Zeile Text dargestellt werden.

Methoden:

- setText(String)
- setFont(new Font("Arial", Font.BOLD,18))
- setVisible

7.3.2 JTextField

Ein JTextField kann eine Eingabe entgegen nehmen. Es gibt kein onChange-Event, nur ein Return-Tasten-Event.

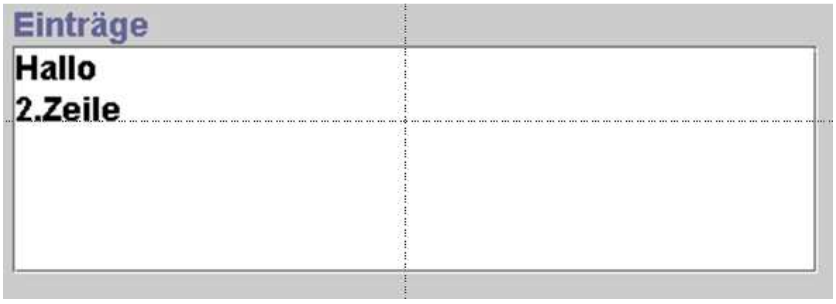


Methoden:

- setText(String)
- String s = getText()
- isEditable()
- isEnabled();
- copy()
- cut()
- paste()
- setFont(new Font("Arial", Font.BOLD,18))
- setVisible
- setEnabled

7.3.3 JTextArea

Ein JTextArea kann mehrere Textzeilen entgegen nehmen. Es gibt kein onChange-Event.



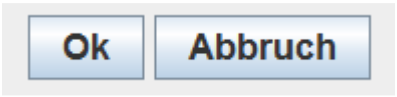
Methoden:

- setText(String)
- String s = getText()
- append(String)

- isEditable()
- isEnabled();
- copy()
- cut()
- paste()
- setFont(new Font("Arial", Font.BOLD,18))
- setVisible
- setEnabled

7.3.4 JButton

Ein JButton dient zum Aktivieren von Aktionen (Save, Load etc.).



Methoden:

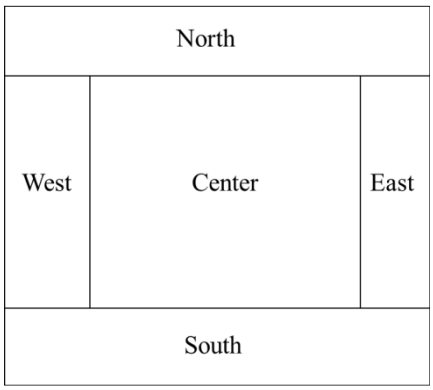
- setText(String)
- isEnabled();
- setFont(new Font("Arial", Font.BOLD,18))
- setVisible
- setEnabled

7.4 Bildschirmgestaltung: Layout

Layout-Verwalter:

- BorderLayout
- GridLayout
- FlowLayout Für Schalter mit einem zusätzlichen JPanel
- CardLayout
- BoxLayout
- GridBagLayout

7.4.1 BorderLayout



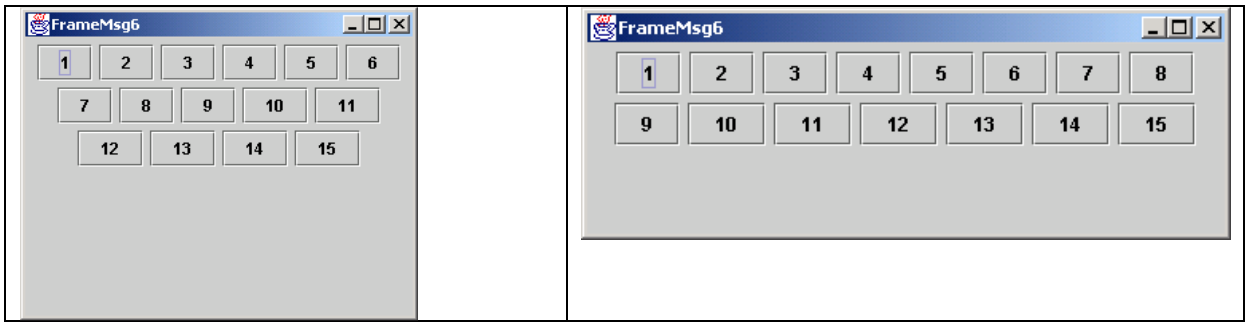

```
private void setGUI() {
   JButton bn1;
   this.getContentPane().setLayout ( new BorderLayout(3,1) );
   bn1 = new JButton("1. Text");  // Schalter erzeugen
   this.getContentPane().add(bn1, BorderLayout.SOUTH);
} // setGUI
```

7.4.2 FlowLayout

Die Elemente sind nicht fest, sondern fließen je nach Breite.

Konstruktor: FlowLayout.RIGHT / CENTER / LEFT

```
private void setGUI() {
   JButton bn;
   this.getContentPane().add( new FlowLayout() );
   for (int i=0; i<15; i++) {
      bn = new JButton( Integer.toString(i+1)  );
      this.getContentPane().add(bn);
   }
} // setGUI
```



Je nach Breite fließen die Schalter

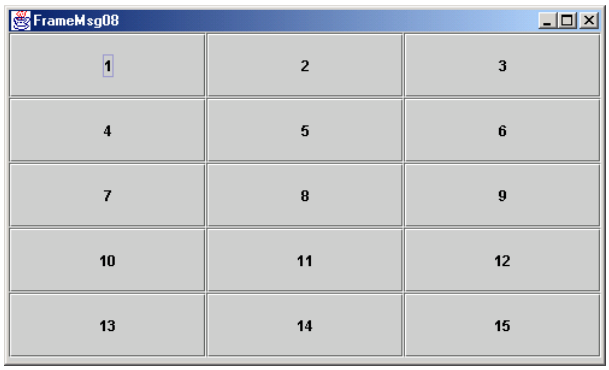
Muster für einen Schalterpanel:

- `JPanel panelBn = new JPanel();`
- `panelBn.setLayout( new FlowLayout( FlowLayout.RIGHT) );`
- `panelBn.add(BnOk);`
- `panelBn.add(BnEsc) ;`
- `this.getContentPane().add(panelBn, BorderLayout.SOUTH);`

7.4.3 GridLayout

Feste Anzahl, Rechteck-Felder

```
private void setGUI() {
   this.getContentPane().setLayout ( new GridLayout(5,3) );
   for (int i=0; i<15; i++) {
      bn = new JButton( Integer.toString(i+1)  );
      this.getContentPane().add(bn);
   }
} // setGUI
```



7.4.4 GridBagLayout

Flexibelstes Layout

Klasse Insets:
Konstruktor:
Insets(int top, int left, int bottom, int right)

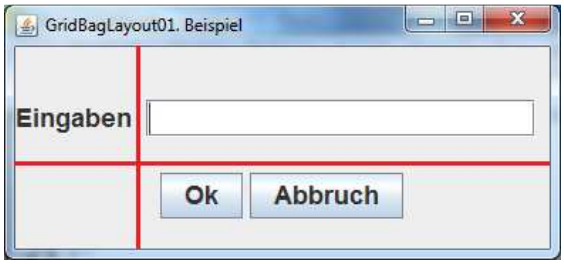
Klasse GridBagContraints:

- int gridx — Element in der gridx-te Spalte
- int gridy — Element in der gridy-te Zeile
- int gridwidth — Anzahl der Spalten
- int gridheight — Anzahl der Zeilen
- double weightx — Verteilung von Platzänderung (Prozentual)
- double weighty — Verteilung zusätzlichen Platzes (Prozentual)
- int anchor — Verknüpfung (NORTHEAST, EAST, SOUTHEAST, CENTER, SOUTH, SOUTHWEST, WEST, NORTH, NORTHWEST)
- int fill — Ausfüllen (NONE, HORIZONTAL, BOTH, VERTICAL)
- Insets insets — externer Lückenbüßer (absolut), äußere Rand, Top, Left, Bottom, Right
- int ipadx — interner Lückenbüßer (absolut), Breite des GUI-Elements
- int ipady — interner Lückenbüßer (absolut), Höhe des GUI-Elements

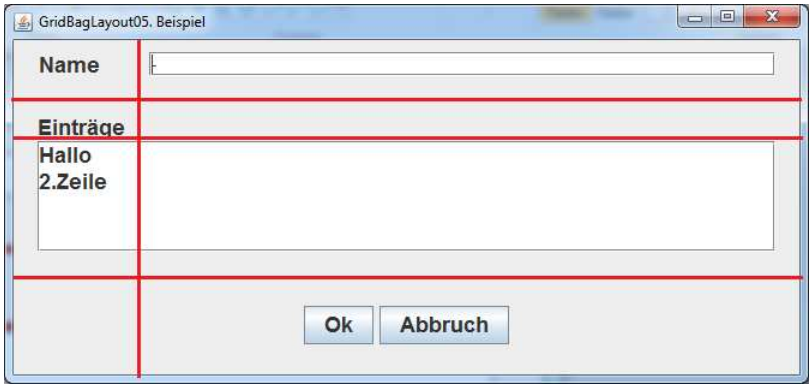
Beispiel:

```
this.getContentPane().add(jLabel1,
    new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,GridBagConstraints.WEST,
            GridBagConstraints.NONE, new Insets(4,20,4,0), 00, 0));
```

```
    this.getContentPane().add(TName,
        new GridBagConstraints(1, 0, 3, 1, 1.0, 0.0,GridBagConstraints.WEST,
                GridBagConstraints.HORIZONTAL, new Insets(4,0,4,20), 00, 0));

   this.getContentPane().add(BnOk,
        new GridBagConstraints(0, 1, 2, 1, 0.0, 0.0,GridBagConstraints.CENTER,
               GridBagConstraints.NONE, new Insets(4,20,4,0), 0, 0));

   this.getContentPane().add(BnESC,
        new GridBagConstraints(2, 1, 1, 1, 0.0, 0.0,GridBagConstraints.CENTER,
              GridBagConstraints.NONE, new Insets(4,20,4,0), 0, 0));
```



Beispiel5:




```
// Beispiel mit GridBagLayout und einem Editor
// weighty  und BOTH

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class GridBagLayout05 extends JFrame {

  JLabel jLabel1 = new JLabel();
  JTextField TName = new JTextField();
  JLabel jLabel2 = new JLabel();
  JTextArea TEdit = new JTextArea("Hallo\n2.Zeile");
  JButton BnOk = new JButton("Ok");
  JButton BnEsc = new JButton("Abbruch");


  public GridBagLayout05() {
    setSize(400, 350);
    setTitle("GridBagLayout05. Beispiel");
    setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    setGUI();
  }
```

```
  private void setGUI() {
    this.getContentPane().setLayout( new GridBagLayout() );

    jLabel1.setText("Name");
    TName.setText("-");
    jLabel2.setText("Einträge");

    BnEsc.addActionListener(new java.awt.event.ActionListener() {

      public void actionPerformed(ActionEvent e) {
        BnEsc_Clicked();
      }
    });


    this.getContentPane().add(jLabel1,
        new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
            ,GridBagConstraints.NORTH,
            GridBagConstraints.NONE,
            new Insets(10,20,30,20), 0, 0));

    this.getContentPane().add(TName,
        new GridBagConstraints(1, 0, 1, 1, 100.0, 0.0
            ,GridBagConstraints.NORTH,
            GridBagConstraints.HORIZONTAL,
            new Insets(10,20,30,20), 0, 0));

    this.getContentPane().add(jLabel2,
        new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
            ,GridBagConstraints.NORTH,
            GridBagConstraints.NONE,
            new Insets(0,20,0,0), 0, 0));

    this.getContentPane().add(
            new JScrollPane(TEdit),
        new GridBagConstraints(
            0, 2, 2, 1, 100.0, 100.0
            ,GridBagConstraints.EAST,
            GridBagConstraints.BOTH,
            new Insets(0,20,0,20), 0, 0));


            // Schalter in einem zusaetzlichem JPanel
     JPanel panelBn = new JPanel();
     panelBn.setLayout( new FlowLayout() );
     //panelBn.setBackground(Color.green);

    this.getContentPane().add(panelBn,
        new GridBagConstraints(0, 3, 2, 1, 100.0, 0.0
            ,GridBagConstraints.CENTER,
            GridBagConstraints.HORIZONTAL,
            new Insets(40,20,20,20), 0, 0));
    panelBn.add(BnOk);
    panelBn.add(BnEsc);


  jLabel1.setFont( new Font("Arial", Font.BOLD,18)  );
  jLabel2.setFont( new Font("Arial", Font.BOLD,18)  );
  TEdit.setFont( new Font("Arial", Font.BOLD,18)  );
  BnOk.setFont( new Font("Arial", Font.BOLD,18)  );
  BnEsc.setFont( new Font("Arial", Font.BOLD,18)  );
 } // setGUI

// Aktion Methoden

  void BnEsc_Clicked() {
    System.exit(0);
  }
```

```
  public static void main(String[] args) {
    GridBagLayout05 frame = new GridBagLayout05();
    frame.setVisible(true);
 }

}
```

7.5 Menüs

```
public class menuBsp extends JFrame {

   public setGUI() {
      menuBar1 = new JMenuBar();
      MainFile = new JMenu("Datei");
      MnNew = new JMenuItem("New File");
      MnNew.setText("Neue Datei");
      MnNew.setAccelerator( KeyStroke.getKeyStroke(KeyEvent.VK_N,
              ActionEvent.CTRL_MASK));

      MnNew.addActionListener(new ActionListener()  {
          public void actionPerformed(ActionEvent e) {
         MnNew_actionPerformed(e);
       }
     });
     MainFile.add(MnNew);
     menuBar1.add(MainFile);
     this.setJMenuBar(menuBar1);
  }
}
```

7.6 ActionListener

7.6.1 Anonyme Klasse

```
private void setGUI() {
   BnOk = new JButton( "Ok" );
   BnOk.addActionListener(  new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
         BnOk_Click(e);
      }
   }
   );
}  // setGUI
```

7.6.2 Eine Methode

```
public class Action5 extends JFrame implements ActionListener {
   JButton BnOk = new JButton("Ok");
   JButton BnEsc = new JButton("Abbruch");

   private void setGUI() {
      BnOk.addActionListener(this);
      BnEsc.addActionListener(this);
   } // setGUI

   // Aufgerufen von allen Schaltern
   public void actionPerformed(ActionEvent e) {
```

```
      System.out.println("Hier actionPerformed");
      if (e.getSource()==BnOk)
         System.out.println("Hier BnOk");
      if (e.getSource()==BnEsc)
         System.out.println("Hier BnEsc");
   }
}  // Action5
```

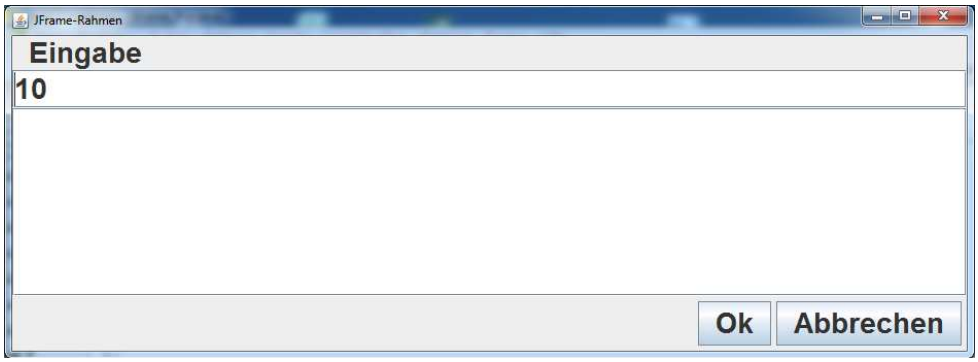
7.6.3 Eine Methode, ohne instanceOf

```
public class Action5 extends JFrame implements ActionListener {
   JButton BnOk = new JButton("Ok");
   JButton BnEsc = new JButton("Abbruch");

   private void setGUI() {
      BnOk.addActionListener(this);
      BnOk.setActionCommand("Ok");

      BnEsc.addActionListener(this);
      BnEsc.setActionCommand("Esc");

   } // setGUI

   public void actionPerformed(ActionEvent e) {
      String cmd = e.getActionCommand();
      if (cmd.equals("Ok") ) {
         BnOk_Click();
      }
      if (cmd.equals("Esc") ) {
         BnEsc_Click();
      }

   }
}  // Action5
```

7.7 JFrame-Beispiel

```
// verwendete Package
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JFrame_Bsp extends JFrame {

  JTextField Editzeile = new JTextField();
  JTextArea Editor = new JTextArea();

  //Frame konstruieren
  public JFrame_Bsp() {
    setSize(1000, 700);
    setTitle("JFrame-Rahmen");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  // wenn close, dann
    setGUI();
  } // create


void setGUI() {
    JLabel Label1 = new JLabel("Eingabe");
    JButton BnOk = new JButton("Ok");     // Schalter definieren
    JButton BnEsc = new JButton("Abbrechen");     // Schalter definieren
    Box box1;  // Container für die Eingabe
```

```
      // North
    Label1.setFont( new Font("Arial", Font.BOLD,28));
    Editzeile.setFont( new Font("Arial", Font.BOLD,28));
    Editzeile.setText("10");  // N =10

    box1 = Box.createVerticalBox();
    //box1.add(  Box.createVerticalStrut(5) );
    box1.add(Label1);
    //box1.add(  Box.createVerticalStrut(5) );
    box1.add(Editzeile) ;
    //box1.add(  Box.createVerticalStrut(30) );
    this.getContentPane().add(box1, BorderLayout.NORTH);

      // Center
    Editor.setFont( new Font("Arial", Font.BOLD,28));
    Editor.setText("");
    getContentPane().add( new JScrollPane(Editor), BorderLayout.CENTER);

      // South
    BnOk.setFont( new Font("Arial", Font.BOLD,28));
    BnEsc.setFont( new Font("Arial", Font.BOLD,28));

    JPanel panelBn = new JPanel();
    panelBn.setLayout( new FlowLayout( FlowLayout.RIGHT) );
    panelBn.add(BnOk);
    panelBn.add(BnEsc) ;
    this.getContentPane().add(panelBn, BorderLayout.SOUTH);

    BnOk.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        BnOk_Click();
      }
    });
    BnEsc.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        BnEsc_Click();
      }
    });

  }  // setGUI


  void BnOk_Click() {
    Editor.setText("Ok_Click");  // Editorinhalt löschen
  }  // BnOk_Click


  void BnEsc_Click() {
    this.dispose();
    System.exit(0);
  }


  public static void main(String[] args) {
    JFrame_Bsp frame = new JFrame_Bsp();
    frame.setVisible(true);
  }

}
```


JFrame-Rahmen
Eingabe
10
Ok
Abbrechen

8 Exception

8.1 Standard Runtime Exceptions:

- ArithmeticException
- ClassCastException
- IllegalArgumentException
- NumberFormatException
- IndexOutOfBoundsException
- NullPointerException

8.2 java.io Exception

- IOException
- EOFException
- FileNotFoundException
- InterruptedIOException
- UTFDataFormatException

Beispiele:

```
try {// Excecption Block
      k=3 / j;
    }
    catch (ArithmeticException f) {
         System.err.println(" ArithmeticException : " + f);
    }
```

```
public static double loadDouble(String sFileName) throws IOException {
   int i=1;
   if (i==1) throw new EOFException("Modul loadDouble (Header hat falsche Version)");
   return 1.234;
 }  // loadDouble
```

8.3 finally

Finally-Blöcke folgen nach einem Catch-Block und werden immer ausgeführt egal, ob im Try-Block ein Fehler aufgetreten ist oder nicht. So können z.B. Datenströme oder Dateien geschlossen werden. Somit muss dies nicht redundant im Try- und Catch-Block geschehen.

```
FileInputStream fin=null;
DataInputStream din=null;
byte b;
try {
   fin = new FileInputStream( sFilename );
   din = new DataInputStream(fin);
   while  ( din.available()>0 ) {
      b  = din.readByte();
      System.out.println((char) (b) );
   }
}
catch ( FileNotFoundException e )  {
   System.err.println( "Die Datei ist nicht vorhanden!" );
}
catch ( IOException e )  {
   System.err.println( "Schreib-/Leseprobleme!" );
}
finally {
   if ( fin != null )
   try { fin.close(); }
      catch ( IOException e ) {
         e.printStackTrace();
   }
}
```

9 Ein- und Ausgabe

9.1 Eingabe

Methoden der Klasse DataInputStream

- readBoolean
- readByte (…) Lesen einer 8-Bit Vorzeichenzahl
- readChar (…) Lesen einer 16-Bit vorzeichenlosen Zahl
- readDouble (…) Lesen einer Double-Zahl
- readFloat (…) Lesen einer Single-Zahl
- readInt (…) Lesen einer 32-Bit Vorzeichenzahl
- readLong (…) Lesen einer 64-Bit Vorzeichenzahl
- readShort (…) Lesen einer 16-Bit Vorzeichenzahl
- readUTF(…) Lesen eines Strings

```
import java.io.*;

private void test1(){
   FileInputStream fin;
   InputStreamReader iin;
   LineNumberReader din;
   String sLine;
   try {
      fin = new FileInputStream("read3.java");
      iin = new InputStreamReader(fin);
      din = new LineNumberReader(iin);
      while  ( din.ready()  ) {
         sLine = din.readLine();
         System.out.println(sLine);
      }
   }
   catch (IOException e) {
      System.err.println("IOException: " + e);
   }
}


private void test2(String sFilename) {
   int i;
   try {
      FileInputStream Fin = new FileInputStream(sFilename);
      DataInputStream DataIn = new DataInputStream(Fin);
      i = DataIn.readInt();
      System.out.println(i);
      i = DataIn.readInt();
      System.out.println(i);
      i = DataIn.readInt();
      System.out.println(i);
      DataIn.close();
   }
   catch (IOException e) { }
   }
}
```

9.2 Ausgabe

Methoden der Klasse DataOutputStream

- writeBoolean (Schreiben eines 8-Bit Logikwertes)
- writeByte (Schreiben einer 8-Bit Vorzeichenzahl)
- writeChar (Schreiben einer 16-Bit vorzeichenlosen Zahl)
- writeDouble (Schreiben einer Double-Zahl)
- writeFloat (Schreiben einer Single-Zahl)
- writeInt (Schreiben einer 32-Bit Vorzeichenzahl)
- writeLong (Schreiben einer 64-Bit Vorzeichenzahl)
- writeShort (Schreiben einer 16-Bit Vorzeichenzahl)
- writeUTF (Schreiben eines Strings)
- writeChars (Schreiben eines char-Arrays oder Strings)

Beispiel:

```
import java.io.*;

public class Ausgabe {

public test1() {
try {
FileOutputStream Fout = new FileOutputStream("1.dat");
PrintStream p = new PrintStream(Fout);
p.print("Hallo");
p.println("Student: "+12345);
p.close();
}
catch (IOException ee) {
System.err.println("IOException: " + ee);
}
}  // test


private test2() {
   double d;
   try {
      FileOutputStream Fout = new FileOutputStream("1.bin");
      DataOutputStream Dout = new DataOutputStream(Fout);
      d = 1234.0;
      Dout.writeDouble(d);
      int i = 1234;
      Dout.writeInteger(i);
      Dout.close();
   }
   catch (IOException e) {
      System.err.println("IOException: " + e);
   }
}  // test
```

9.3 Read/Write mit RandomAccessFile

Lesen an beliebiger Stelle (zum Beispiel HexEditor).

Methoden:

- int read(byte[] b, int off, int len)
- boolean readBoolean()
- byte readByte()
- char readChar()

- double readDouble()
- float readFloat()
- int readInt()
- String readLine()
- long readLong()
- short readShort()
- String readUTF()
- int readUnsignedByte()
- int readUnsignedShort()
- void setLength(long newLength)
- int skipBytes(int n)
- length Ausgabe der Länge der Datei
- void seek(long pos)
- void setLength(long newLength)
- int **skipBytes**(int n)
- void readFully(byte[] b)
  Reads b.length bytes from this file into the byte array,
  starting at the current file pointer.
- void readFully(byte[] b, int off, int len)
  Reads exactly len bytes from this file into the byte array,
  starting at the current file pointer.

Beispiel:

```
private byte[] read(int count, int offset, String sFilename) {
   RandomAccessFile fin =  new RandomAccessFile( sFilename, "r");
   if (offset>0) fin.seek(offset);

   byte [] feld = new byte[count];
        int anz = fin.read(feld, 0, count);
        fin.close();
   return feld;
}
```

10 Besondere Funktionen

10.1 Rahmen um GUI-Elemente

Mit der Border-Klasse kann man um einzelne oder mehrere GUI-Elemente spezielle Rahmen zeichnen lassen.

```
import javax.swing.border.*;
TMatrNr.setText("12741");
tb = new TitledBorder( LineBorder.createBlackLineBorder(), "Matr.-Nr");
TMatrNr.setBorder(tb);
```

10.2 ClipBoard

```
// Allgemeine Funktionen
// für die Zwischenablage Clipboard
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.ClipboardOwner;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.StringSelection;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.UnsupportedFlavorException;


StringSelection stringSelection = new StringSelection( myString );
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    clipboard.setContents( stringSelection, this );
```

Zusätzlich:

```
implements ClipboardOwner
public void lostOwnership( Clipboard aClipboard, Transferable aContents) {
     //do nothing
   }
```

10.3 Look & Feel

Mit dem Look and Feel kann das Aussehen des Programms geändert werden:

```
public void MnLook_and_Feel(ActionEvent e) {
  try {
   if (e.getSource() == MnLook_Feel_Windows ) {
    UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    SwingUtilities.updateComponentTreeUI(this.getContentPane() );
  }
  if (e.getSource() == MnLook_Feel_Metal ) {
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
    SwingUtilities.updateComponentTreeUI(this.getContentPane() );
  }
  if (e.getSource() == MnLook_Feel_Motif ) {
   UIManager.setLookAndFeel(
```

```
   "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
   SwingUtilities.updateComponentTreeUI(this.getContentPane() );
  }
  if (e.getSource() == MnLook_Feel_Macintosh ) {
   UIManager.setLookAndFeel("javax.swing.plaf.mac.MacLookAndFeel");
   SwingUtilities.updateComponentTreeUI(this.getContentPane() );
  }
  } catch (Exception exc) {
}
```

10.4 Export in eine Excel-Datei

Mit dem Binary Interchange FileFormat kann man direkt eine Excel-Datei erstellen. Die Klasse Export2Excel kapselt die kompletten Details

Rahmen:

- Export2BIFF excel = new Export2BIFF("test2.xls");
- excel.openFile();
- excel.Write(1,2,1234); // col row Value
- excel.Write(1,3,2004); // col row Value
- excel.closeFile();

```
import java.io.*;

private void test_Excel() {
        int row;
        Export2BIFF excel = new Export2BIFF("test4.xls");
        excel.openFile();
        java.util.Random random = new java.util.Random();
        excel.Write(1,0, "Name" );
        excel.Write(2,0, "Integer" );
        excel.Write(3,0, "Gehälter" );
        for (row=1; row<50; row++) {
          excel.Write(1,row, "abc" + Integer.toString(row) );  // col row Value
          excel.Write(2,row, random.nextInt() );
          excel.Write(3,row, random.nextDouble()*10000 );
        } // for
        excel.closeFile();
        StartExcel();
    }

    private void StartExcel() {
    try {
      // funktioniert nur mit vollstaendigen Pfad für das Excel-Programm,
      Runtime.getRuntime().exec("excel.exe   D:\\Excel\\test.xls");
    }
    catch (IOException e1) {
       System.out.println("Error in Start Excel");
    }
}
```

10.4.1 Export2BIFF

```
/*

   Autor:           Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
   Firma:           HS Harz, FB AI
   Version:         1.1
   Ersterstellung:  09.11.2010
```

```
   Letzte Änderung: 09.11.2010
   EWrstellt einen nativen Export nach Excel, ohne csv

  Export nach Excel mittels des Binary Interchange File Format (BIFF)
  Hauptproblem:  Windows speichert little Endian
                 Java    speichert big Endian

*/

// import java.io.*;

class Export2BIFF{

   // interne Variablen fuer das Speichern
  private FileOutputStream _fout;
  private DataOutputStream _dout;
  private String _sFilename;

    // Felder  Beginn Ende  String  Int
  byte [] CXlsBof     = { 9, 8, 8, 0, 0, 0, 0x10, 0, 1, 0, 0, 0 };
  byte [] CXlsEof     = {0x0A, 0, 0, 0};
  byte [] CXlsString  = new byte[12];
  byte [] CXlsDouble  = new byte[10];
  byte [] CXlsInt     = new byte[10];


  public Export2BIFF(String sFilename) {
    _sFilename = sFilename;
    ClearArrays();
  }  // Export2BIFF


  public void openFile() {
   try {
    _fout = new FileOutputStream(_sFilename);
    _dout = new DataOutputStream(_fout);
    for (int i=0; i<CXlsBof.length; i++) {
      _dout.writeByte(CXlsBof[i]);
    }
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
  }  // Export2Modul


  public void closeFile() {
   try {
    for (int i=0; i<CXlsEof.length; i++) {
      _dout.writeByte(CXlsEof[i]);
    }
    _dout.close();
    _fout.close();
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
  }  // close()


  public void ClearArrays() {
      // eigentlich sind das Word, unsigned short, 16 Bit, die Reihenfolge passt
aber nicht
      // deshalb byteweise
     CXlsString[0] =  4;
     CXlsString[1] =  2;
     CXlsString[2] =  0;
     CXlsString[3] =  0;
     CXlsString[4] =  0;
     CXlsString[5] =  0;
```

```
     CXlsString[6] =  0;
     CXlsString[7] =  0;
     CXlsString[8] =  0;
     CXlsString[9] =  0;
     CXlsString[10]=  0;
     CXlsString[11]=  0;

      // eigentlich sind das Word, unsigned short, 16 Bit, die Reihgenfolge passt
aber nicht
      // deshalb byteweise
     CXlsDouble[0] = 3;
     CXlsDouble[1] = 2;
     CXlsDouble[2] = 14;
     CXlsDouble[3] = 0;
     CXlsDouble[4] = 0;
     CXlsDouble[5] = 0;
     CXlsDouble[6] = 0;
     CXlsDouble[7] = 0;
     CXlsDouble[8] = 0;
     CXlsDouble[9] = 0;

      // eigentlich sind das Word, unsigned short, 16 Bit, die Reihgenfolge passt
aber nicht
      // deshalb byteweise
     CXlsInt[0] = 0x7E;
     CXlsInt[1] = 2;
     CXlsInt[2] = 10;
     CXlsInt[3] = 0;
     CXlsInt[4] = 0;
     CXlsInt[5] = 0;
     CXlsInt[6] = 0;
     CXlsInt[7] = 0;
     CXlsInt[8] = 0;
     CXlsInt[9] = 0;
  }  // ClearArrays



   // schreiben der Vorspann eines Int
  private void writeCXlsInt() {
   try {
    for (int i=0; i<CXlsInt.length; i++) {
      _dout.writeByte(CXlsInt[i]);
    }
   }
   catch (IOException e) {
         System.err.println("IOException: " + e);
   }
  }


  public void Write(int Col, int Row, int iValue) {
     short s;
     int v;
     byte i3,i2,i1,i0;
     try {
        ClearArrays();
        s = (short) (Row);
        CXlsInt[4] = (byte) (s & 255);  // 4 5   [2]
        CXlsInt[5] = (byte) ((s >> 8) & 255);  // 4 5   [2]

        s = (short) (Col);
        CXlsInt[6] = (byte) (s & 255);  // 6 7 [3]
        CXlsInt[7] = (byte) ((s >> 8) & 255);  // 6 7 [3]
        writeCXlsInt();

          // Trennung der Bytes, à Resim
        v = (iValue << 2) | 2;
        i0 = (byte) (v & 255);
```

```
        i1 = (byte) ((v >> 8) & 255);
        i2 = (byte) ((v >> 16) & 255);
        i3 = (byte) ((v >> 24) & 255);
        _dout.writeByte( i0 );
        _dout.writeByte( i1 );
        _dout.writeByte( i2 );
        _dout.writeByte( i3 );
     }
     catch (IOException e) {
         System.err.println("IOException: " + e);
     }
  }  // Write Int


   // schreiben der Vorspann eines double
  private void writeCXlsDouble() {
   try {
    for (int i=0; i<CXlsDouble.length; i++) {
      _dout.writeByte(CXlsDouble[i]);
    }
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
  }


  // ein double-Wert wird genau verkehrt herum in Java geschrieben
  // also aufsplitten der einzelnen Bytes
  public void Write(int Col, int Row, double dValue) {
     short s;
     int v;
     int i;
     int [] bits = new int[8];  // int statt byte, da byte Vorzeichenbehaftig

     try {
        ClearArrays();
        s = (short) (Row);
        CXlsDouble[4] = (byte) (s & 255);  // 4 5   [2]
        CXlsDouble[5] = (byte) ((s >> 8) & 255);  // 4 5   [2]

        s = (short) (Col);
        CXlsDouble[6] = (byte) (s & 255);  // 6 7 [3]
        CXlsDouble[7] = (byte) ((s >> 8) & 255);  // 6 7 [3]
        writeCXlsDouble();

         // Konvertierung in long, auftrennen der einzelnen Bytes der ACHT Double-
Bytes !!!!
         // Tja, sowas gibt es in jaaaava
       long lD = Double.doubleToLongBits(dValue);

       int ofs;
       for (i=0, ofs=0; i<bits.length; i++, ofs+=8) {
         bits[i] = (byte) ((lD >> ofs) & 255);   // GDI 1 Resim
       }


        for (i=0; i<bits.length; i++) {
          _dout.writeByte( bits[i] );
          if (bits[i]<0)
           System.out.println("i: "+i+"   "+(int) (bits[i]+256));
          else
          System.out.println("i: "+i+"   "+bits[i]);
        }
        // _dout.writeDouble( Value );  // leider hier verkehrte Reihenfolge
     }
     catch (IOException e) {
         System.err.println("IOException: " + e);
     }
  }  // Write Double
```

```
   // schreiben der Vorspann eines String
  private void writeCXlsString() {
   try {
    for (int i=0; i<CXlsString.length; i++) {
      _dout.writeByte(CXlsString[i]);
    }
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
  }


  public void Write(int Col, int Row, String sValue) {
   short len;
   short s;
   try {
     ClearArrays();
     len = (short) sValue.length() ;

     s = (short) (len+8);
     CXlsString[2] = (byte) (s & 255);  // 2 3   [1]   erst low dann high byte
     CXlsString[3] = (byte) ((s >> 8) & 255);  // 2 3


     s = (short) (Row);
     CXlsString[4] = (byte) (s & 255);  // 4 5   [2]
     CXlsString[5] = (byte) ((s >> 8) & 255);  // 4 5   [2]

     s = (short) (Col);
     CXlsString[6] = (byte) (s & 255);  // 6 7 [3]
     CXlsString[7] = (byte) ((s >> 8) & 255);  // 6 7 [3]

     s = (short) (len);
     CXlsString[10] = (byte) (s & 255);  // 10 11  [5]
     CXlsString[11] = (byte) ((s >> 8) & 255);  // 10 11  [5]
     writeCXlsString();
     for (int i=0; i<len; i++) {
      _dout.writeByte( (byte) sValue.charAt(i) );  // cahr ist unicopde, zwei Byte
!
     }
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
  }  // Write string



  // alle diese Konstanten sind nicht benutzt
   private final byte  BIFF12_DEFINEDNAME            = 0x27;
   private final short BIFF12_FILEVERSION            = 0x0180;
   private final short BIFF12_WORKBOOK               = 0x0183;
   private final short BIFF12_WORKBOOK_END           = 0x0184;
   private final short BIFF12_BOOKVIEWS              = 0x0187;
   private final short BIFF12_BOOKVIEWS_END          = 0x0188;
   private final short BIFF12_SHEETS                 = 0x018F;
   private final short BIFF12_SHEETS_END             = 0x0190;
   private final short BIFF12_WORKBOOKPR             = 0x0199;
   private final short BIFF12_SHEET                  = 0x019C;
   private final short BIFF12_CALCPR                 = 0x019D;
   private final short BIFF12_WORKBOOKVIEW           = 0x019E;
   private final short BIFF12_EXTERNALREFERENCES     = 0x02E1;
   private final short BIFF12_EXTERNALREFERENCES_END = 0x02E2;
   private final short BIFF12_EXTERNALREFERENCE       = 0x02E3;
   private final short BIFF12_WEBPUBLISHING          = 0x04A9;

   // Worksheet records
```

```
private final byte BIFF12_ROW             = 0x00;
private final byte BIFF12_BLANK           = 0x01;
private final byte BIFF12_NUM             = 0x02;
private final byte BIFF12_BOOLERR         = 0x03;
private final byte BIFF12_BOOL            = 0x04;
private final byte BIFF12_FLOAT           = 0x05;
private final byte BIFF12_STRING          = 0x07;
private final byte BIFF12_FORMULA_STRING  = 0x08;
private final byte BIFF12_FORMULA_FLOAT   = 0x09;
private final byte BIFF12_FORMULA_BOOL    = 0x0A;
private final byte BIFF12_FORMULA_BOOLERR = 0x0B;
private final byte BIFF12_COL             = 0x3C;
private final short BIFF12_WORKSHEET       = 0x0181;
private final short BIFF12_WORKSHEET_END   = 0x0182;
private final short BIFF12_SHEETVIEWS      = 0x0185;
private final short BIFF12_SHEETVIEWS_END  = 0x0186;
private final short BIFF12_SHEETVIEW       = 0x0189;
private final short BIFF12_SHEETVIEW_END   = 0x018A;
private final short BIFF12_SHEETDATA       = 0x0191;
private final short BIFF12_SHEETDATA_END   = 0x0192;
private final short BIFF12_SHEETPR         = 0x0193;
private final short BIFF12_DIMENSION       = 0x0194;
private final short BIFF12_SELECTION       = 0x0198;
private final short BIFF12_COLS            = 0x0386;
private final short BIFF12_COLS_END        = 0x0387;
private final short BIFF12_CONDITIONALFORMATTING = 0x03CD;
private final short BIFF12_CONDITIONALFORMATTING_END = 0x03CE;
private final short BIFF12_CFRULE          = 0x03CF;
private final short BIFF12_CFRULE_END      = 0x03D0;
private final short BIFF12_ICONSET         = 0x03D1;
private final short BIFF12_ICONSET_END     = 0x03D2;
private final short BIFF12_DATABAR         = 0x03D3;
private final short BIFF12_DATABAR_END     = 0x03D4;
private final short BIFF12_COLORSCALE      = 0x03D5;
private final short BIFF12_COLORSCALE_END  = 0x03D6;
private final short BIFF12_CFVO            = 0x03D7;
private final short BIFF12_PAGEMARGINS     = 0x03DC;
private final short BIFF12_PRINTOPTIONS    = 0x03DD;
private final short BIFF12_PAGESETUP       = 0x03DE;
private final short BIFF12_HEADERFOOTER    = 0x03DF;
private final short BIFF12_SHEETFORMATPR   = 0x03E5;
private final short BIFF12_HYPERLINK       = 0x03EE;
private final short BIFF12_DRAWING         = 0x04A6;
private final short BIFF12_LEGACYDRAWING   = 0x04A7;
private final short BIFF12_COLOR           = 0x04B4;
private final short BIFF12_OLEOBJECTS      = 0x04FE;
private final short BIFF12_OLEOBJECT       = 0x04FF;
private final short BIFF12_OLEOBJECTS_END  = 0x0580;
private final short BIFF12_TABLEPARTS      = 0x0594;
private final short BIFF12_TABLEPART       = 0x0595;
private final short BIFF12_TABLEPARTS_END  = 0x0596;

//SharedStrings records
private final byte BIFF12_SI              = 0x13;
private final short BIFF12_SST             = 0x019F;
private final short BIFF12_SST_END         = 0x01A0;

//Styles records
private final byte BIFF12_FONT            = 0x2B;
private final byte BIFF12_FILL            = 0x2D;
private final byte BIFF12_BORDER          = 0x2E;
private final byte BIFF12_XF              = 0x2F;
private final byte BIFF12_CELLSTYLE       = 0x30;
private final short BIFF12_STYLESHEET      = 0x0296;
private final short BIFF12_STYLESHEET_END  = 0x0297;
private final short BIFF12_COLORS          = 0x03D9;
private final short BIFF12_COLORS_END      = 0x03DA;
private final short BIFF12_DXFS            = 0x03F9;
private final short BIFF12_DXFS_END        = 0x03FA;
private final short BIFF12_TABLESTYLES     = 0x03FC;
```

```
    private final short BIFF12_TABLESTYLES_END = 0x03FD;
    private final short BIFF12_FILLS           = 0x04DB;
    private final short BIFF12_FILLS_END       = 0x04DC;
    private final short BIFF12_FONTS           = 0x04E3;
    private final short BIFF12_FONTS_END       = 0x04E4;
    private final short BIFF12_BORDERS         = 0x04E5;
    private final short BIFF12_BORDERS_END     = 0x04E6;
    private final short BIFF12_CELLXFS         = 0x04E9;
    private final short BIFF12_CELLXFS_END     = 0x04EA;
    private final short BIFF12_CELLSTYLES      = 0x04EB;
    private final short BIFF12_CELLSTYLES_END  = 0x04EC;
    private final short BIFF12_CELLSTYLEXFS    = 0x04F2;
    private final short BIFF12_CELLSTYLEXFS_END = 0x04F3;

    //Comment records
    private final short BIFF12_COMMENTS        = 0x04F4;
    private final short BIFF12_COMMENTS_END    = 0x04F5;
    private final short BIFF12_AUTHORS         = 0x04F6;
    private final short BIFF12_AUTHORS_END     = 0x04F7;
    private final short BIFF12_AUTHOR          = 0x04F8;
    private final short BIFF12_COMMENTLIST     = 0x04F9;
    private final short BIFF12_COMMENTLIST_END = 0x04FA;
    private final short BIFF12_COMMENT         = 0x04FB;
    private final short BIFF12_COMMENT_END     = 0x04FC;
    private final short BIFF12_TEXT            = 0x04FD;

    //Table records
    private final short BIFF12_AUTOFILTER      = 0x01A1;
    private final short BIFF12_AUTOFILTER_END  = 0x01A2;
    private final short BIFF12_FILTERCOLUMN    = 0x01A3;
    private final short BIFF12_FILTERCOLUMN_END= 0x01A4;
    private final short BIFF12_FILTERS         = 0x01A5;
    private final short BIFF12_FILTERS_END     = 0x01A6;
    private final short BIFF12_FILTER          = 0x01A7;
    private final short BIFF12_TABLE           = 0x02D7;
    private final short BIFF12_TABLE_END       = 0x02D8;
    private final short BIFF12_TABLECOLUMNS    = 0x02D9;
    private final short BIFF12_TABLECOLUMNS_END= 0x02DA;
    private final short BIFF12_TABLECOLUMN     = 0x02DB;
    private final short BIFF12_TABLECOLUMN_END = 0x02DC;
    private final short BIFF12_TABLESTYLEINFO  = 0x0481;
    private final short BIFF12_SORTSTATE       = 0x0492;
    private final short BIFF12_SORTCONDITION   = 0x0494;
    private final short BIFF12_SORTSTATE_END   = 0x0495;

    //QueryTable records
    private final short BIFF12_QUERYTABLE            = 0x03BF;
    private final short BIFF12_QUERYTABLE_END        = 0x03C0;
    private final short BIFF12_QUERYTABLEREFRESH     = 0x03C1;
    private final short BIFF12_QUERYTABLEREFRESH_END = 0x03C2;
    private final short BIFF12_QUERYTABLEFIELDS      = 0x03C7;
    private final short BIFF12_QUERYTABLEFIELDS_END  = 0x03C8;
    private final short BIFF12_QUERYTABLEFIELD       = 0x03C9;
    private final short BIFF12_QUERYTABLEFIELD_END   = 0x03CA;

    //Connection records
    private final short BIFF12_CONNECTIONS           = 0x03AD;
    private final short BIFF12_CONNECTIONS_END       = 0x03AE;
    private final short BIFF12_CONNECTION            = 0x01C9;
    private final short BIFF12_CONNECTION_END        = 0x01CA;
    private final short BIFF12_DBPR                  = 0x01CB;
    private final short BIFF12_DBPR_END              = 0x01CC;

}
```

11 Indexverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

V

W

Z