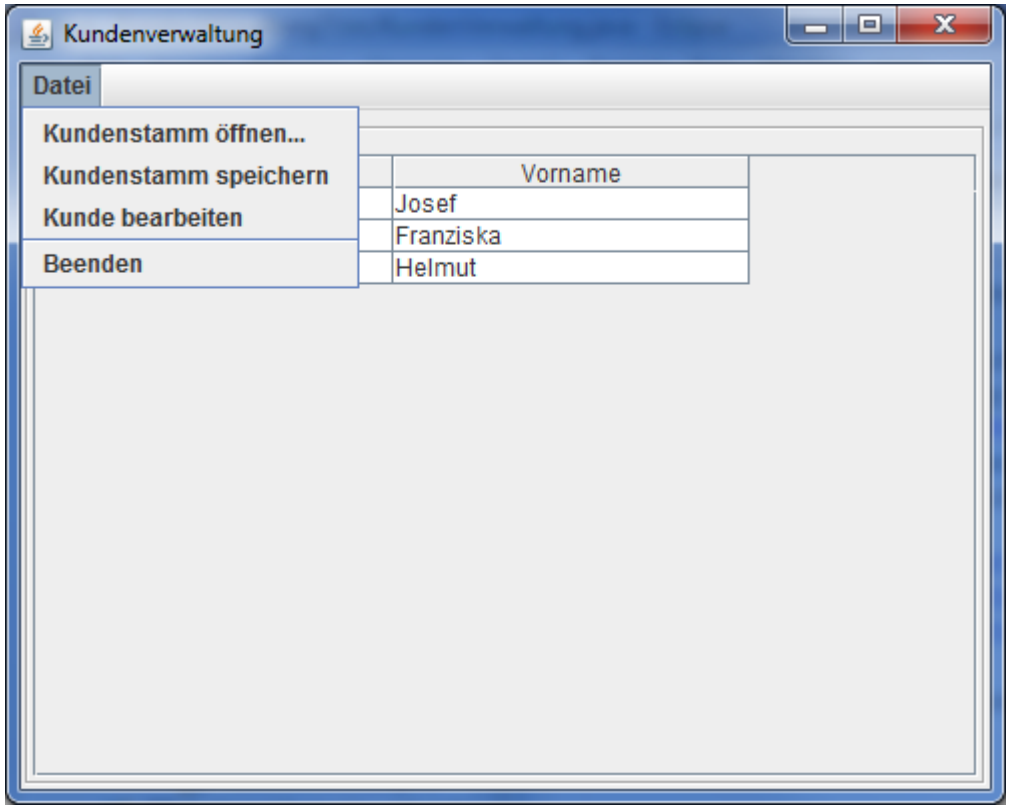
Kapitel 6: Persistenz

Definition

- Persistenz meint den Umgang mit Daten zu deren dauerhafte Speicherung.
- Daten können z.B. im Dateisystem in *flachen* Dateien oder in Datenbanken persistiert werden.

KundenVerwaltung v7

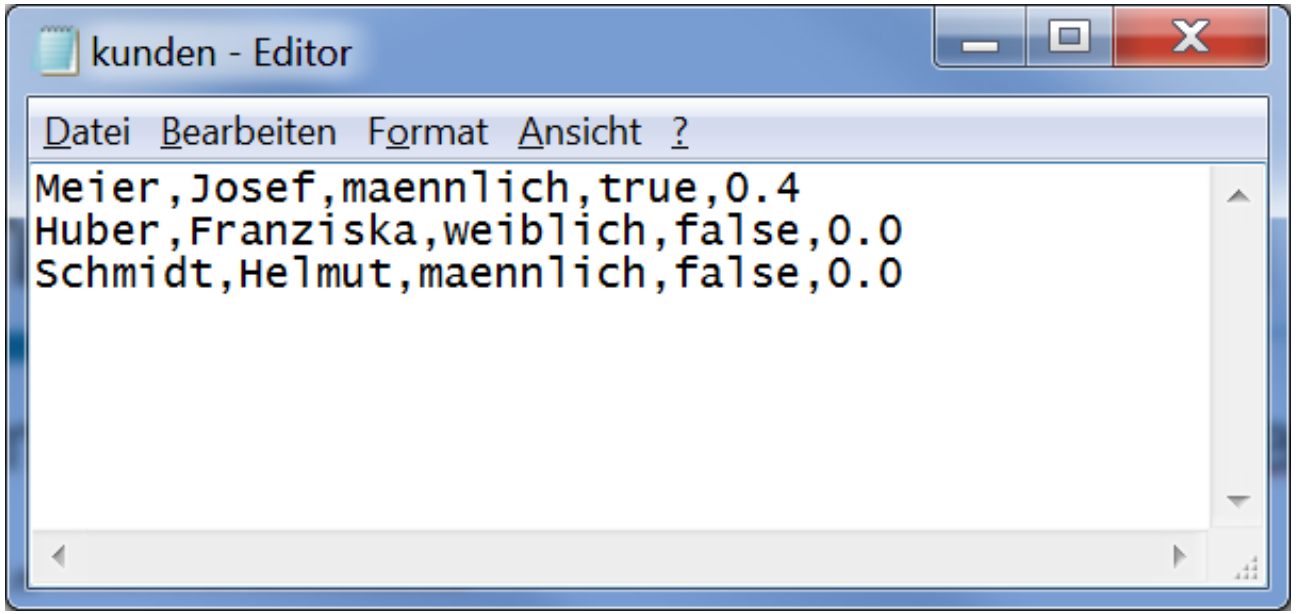


Neue Features:

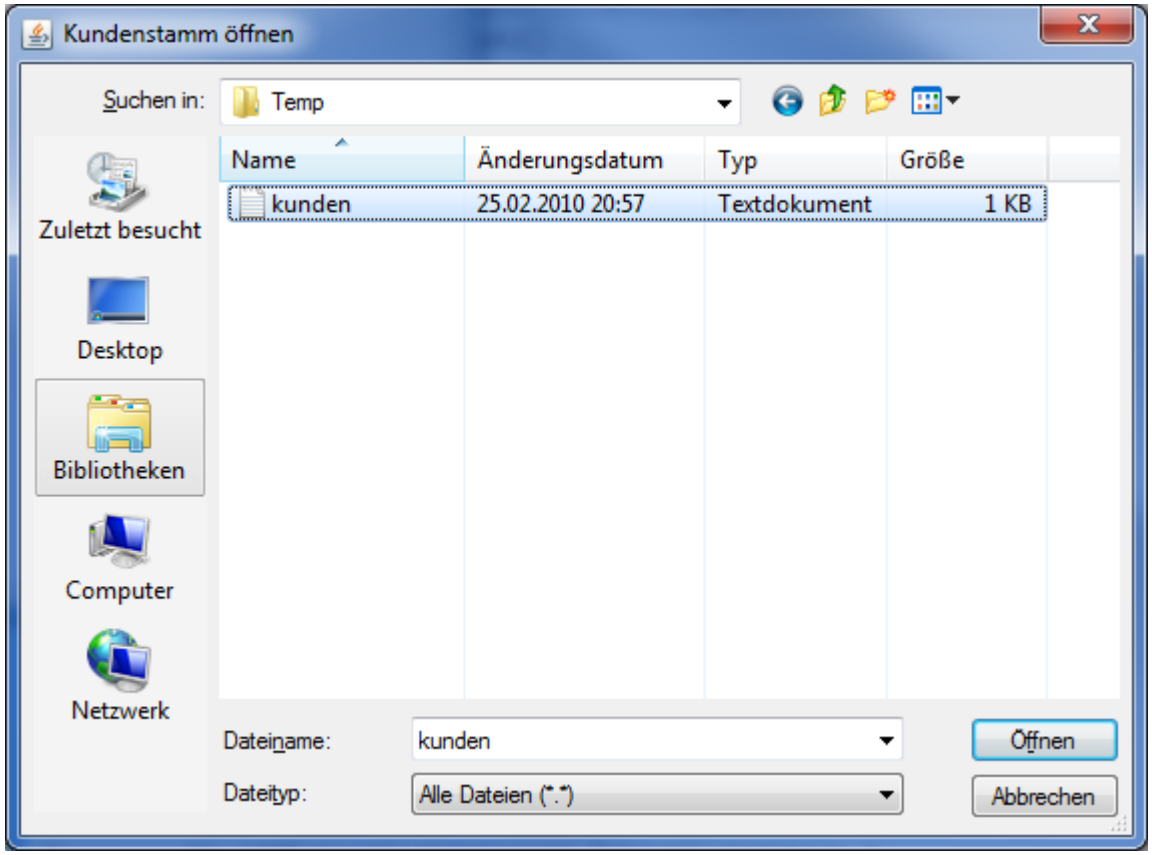
- Laden der Kundendaten per Öffnen-Dialog aus einer Datei
- Speichern der Daten in eine Datei

Kundendaten als CSV-Datei

- Dateiformat: CSV (Comma-Separated-Value)
- Beispiel „kunden.txt“:



Öffnen-Dialog



Datenströme

- Ein Datenstrom (Stream) ist eine Sequenz von Daten, die typischerweise nach dem FIFO-Prinzip verarbeitet wird.
- Das JDK stellt für die Verarbeitung von Datenströmen das Package `java.io` zur Verfügung.

Byteorientierte Datenströme

- Für die Ein- und Ausgabe von Daten enthält `java.io` die abstrakten Klassen `InputStream` (Dateneingabe) und `OutputStream` (Datenausgabe) und deren abgeleitete Klassen.
- Stream-Klassen arbeiten byteorientiert, sie berücksichtigen den vorliegenden Zeichensatz nicht und nehmen ggf. keine Konvertierungen vor.

Zeichenorientierte Datenströme

- Java verwendet zur internen Darstellung Unicode.
- Dateien liegen häufig in ASCII-Code vor.
- Um Daten in Unicode – aus Java-Sicht: zeichenorientiert – verarbeiten zu können, sind in `java.io` die abstrakten Klassen `Reader` und `Writer` sowie deren Derivate enthalten.
- Für die Kodierung bzw. Dekodierung von byteorientierten Input- und Output-Streams stehen die Klassen `InputStreamReader` und `OutputStreamWriter` zur Verfügung.

Übersicht zu `java.io`

Byteorientierte Klassen	Zeichenorientierte Klassen	Beschreibung
`FileInputStream`	`FileReader`	Lesen von und Schreiben in Dateien
`FileOutputStream`	`FileWriter`	
`BufferedInputStream`	`BufferedReader`	Gepufferte Datenein- und ausgabe; Bei der Ausgabe kann mit `flush()` der Puffer explizit weggeschrieben und geleert werden
`BufferedOutputStream`	`BufferedWriter`	
`PrintStream`	`PrintWriter`	Formierte Ausgabe von Text
`ObjectInputStream`		Zur Serialisierung und Deserialisierung von Objekten; Diese müssen `Serializable` implementieren.
`ObjectOutputStream`		

KundeFileManager

```
public class KundeFileManager {

    private String fileName;

    public KundeFileManager(String filename) {
        fileName = filename;
    }

    public List<Kunde> load() throws IOException {
        …
    }

    public void save(List<Kunde> kunden) throws IOException {
        …
    }
}
```

KundeFileManager.load()

```
public class KundeFileManager { …
   public List<Kunde> load() throws IOException {
      BufferedReader br = new BufferedReader(new FileReader(fileName));
      StreamTokenizer st = new StreamTokenizer(br);
      st.whitespaceChars(',', ',');

      List<Kunde> kunden = new ArrayList<Kunde>();

      while (st.nextToken() != StreamTokenizer.TT_EOF) {
         Kunde kunde = new Kunde();
         kunde.setNachname(st.sval);
         st.nextToken(); kunde.setVorname(st.sval);
         st.nextToken(); kunde.setGeschlecht(Geschlecht.valueOf(st.sval));
         st.nextToken(); kunde.setStammkunde(Boolean.valueOf(st.sval));
         st.nextToken(); kunde.setRabattSatz(st.nval);
         kunden.add(kunde);
      }

      br.close();
      return kunden;
   } …
}
```

`StreamTokenizer`

- Mit Hilfe der Klasse `StreamTokenizer` kann ein Eingabestrom gelesen und in sog. *Tokens* zerlegt werden.
- Im Beispiel werden Tokens durch Kommas getrennt (`whitespaceChars()`) und (standardmäßig) durch Zeilenumbrüche.
- Der letzte Token der Datei ist vom Typ `StreamTokenizer.TT_EOF`.
- Stringtokens können mittels `sval` und Zahlentokens (Typ `double`) können mittels `nval` ausgelesen werden.

KundeFileManager.save()

```
public class KundeFileManager { …
   public void save(List<Kunde> kunden) throws IOException {
      FileWriter fw = new FileWriter(fileName);
      BufferedWriter bw = new BufferedWriter(fw);
      PrintWriter pw = new PrintWriter(bw);

      for (Kunde kunde : kunden) {
         pw.print(kunde.getNachname());
         pw.print(',');
         pw.print(kunde.getVorname());
         pw.print(',');
         pw.print(kunde.getGeschlecht());
         pw.print(',');
         pw.print(kunde.isStammkunde());
         pw.print(',');
         pw.println(kunde.getRabattSatz());
      }

      pw.close();
   }
}
```

Streams stets frühest möglich schließen, um unbenötigte Ressourcen freizugeben.

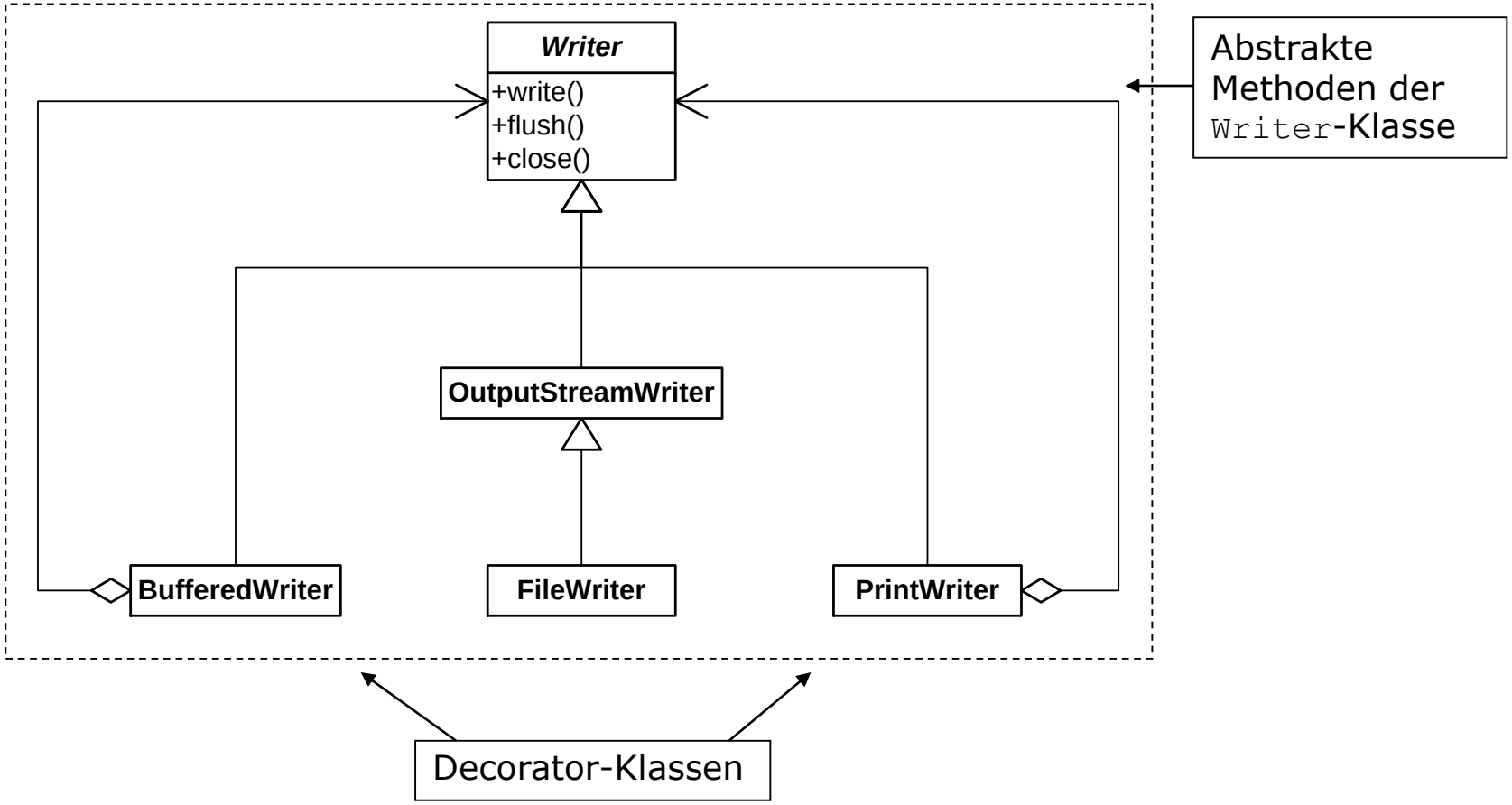
[bs] Decorator-Pattern

- Wiederholung:

> Fügt einem Objekt dynamisch zusätzliche Verantwortlichkeiten hinzu. Decorators bieten eine flexible Alternative zur Vererbung zum Zweck der Erweiterung der Funktionalität.

- `BufferedWriter` und `PrintWriter` dekorieren `Writer`-Klassen und erweitern somit ihre Funktionalität um Pufferung bzw. Formatierungsmöglichkeiten.
- Zu diesem Zweck erben auch Decorators von der Klasse `Writer`.
- Sie besitzen zudem eine Oberklassen-Referenz auf den zu dekorierenden `Writer`.
- Sie überschreiben Methoden der Oberklasse `Writer` und implementieren so ihren *Mehrwert*. Üblicherweise delegieren sie in den überschriebenen Methoden auch an die Methoden der dekorierten Instanz.

[bs] Decorator-Pattern



Standard-Ein- und -Ausgabe

- Die Klasse `System` besitzt folgende drei statische Attribute zu Standard-Ein- und -Ausgabe:

`PrintStream err`	Standard-Fehler-Ausgabestrom
`InputStream in`	Standard-Eingabestrom
`PrintStream out`	Standard-Ausgabestrom

- Die Standard-IO-Streams können auch überschrieben werden, mittels: `setErr()`, `setIn()` und `setOut()`.

Scanner

- Die Klasse `Scanner` existiert seit Java 1.5 und kann als Alternative zur Klasse `StreamTokenizer` gesehen werden.
- Im folgenden Beispiel wird sie zur Verarbeitung von Konsoleneingaben verwendet.

Beispiel

```
public static void main(String[] args) throws Exception {

   FileOutputStream fos =
      new FileOutputStream("err.txt", true);
   PrintStream ps = new PrintStream(fos);
   System.setErr(ps);

   Scanner sc = new Scanner(System.in);

   System.out.print("Zahl: ");
   int x = sc.nextInt();
   System.out.printf("Quadrat: %d", x * x);

   ps.close();
   sc.close();
}
```

Datei für Exceptions zum *Anhängen* öffnen

Standard-Fehler-Ausgabestrom neu setzen

Standard-Eingabestrom mit Scanner verarbeiten

`File`

- Stellt eine abstrakte Repräsentation von Datei- und Verzeichnisnamen dar.
- Die Klasse `File` selbst dient **nicht** zum Lesen aus oder Schreiben in Dateien – dies unterstützen stattdessen Datenströme.

Beispiel

```
private static int findFiles(File file, String extension) {
    if (file.isHidden()) return 0;

    if (file.isFile() && file.getName().endsWith(extension)){
        System.out.println(file.getPath());
        return 1;
    }

    if (!file.isDirectory() || file.listFiles() == null) return 0;

    int count = 0;

    for (File subfile : file.listFiles()) {
        count += findFiles(subfile, extension);
    }

    return count;
}

public static void main(String[] args) {
    File file = new File("C:\\Users");
    int count = findFiles(file, ".docx");
    System.out.println("Anzahl der gefundenen Dateien: " + count);
}
```

Auflisten aller Docx-Dateien im Verzeichnis „C:\Users“

Anbinden des KundeFileManagers

```
public class KundenVerwaltung extends JFrame { …
   private KundeFileManager kundeFileManager;

   public KundenVerwaltung() { …
      menu.add(mi = new JMenuItem("Kundenstamm öffnen..."));
      mi.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
            FileDialog fd = new FileDialog(KundenVerwaltung.this,
               "Kundenstamm öffnen", FileDialog.LOAD);
            fd.setDirectory(".");
            fd.setVisible(true);
            try {
               String filename = fd.getDirectory() + fd.getFile();
               kundeFileManager = new KundeFileManager(filename);
               kundeTableModel.setKunden(kundeFileManager.load());
               kundeTableModel.fireTableDataChanged();
            }
            catch (Exception ex) {}
         }
      }); …
   } …
}
```

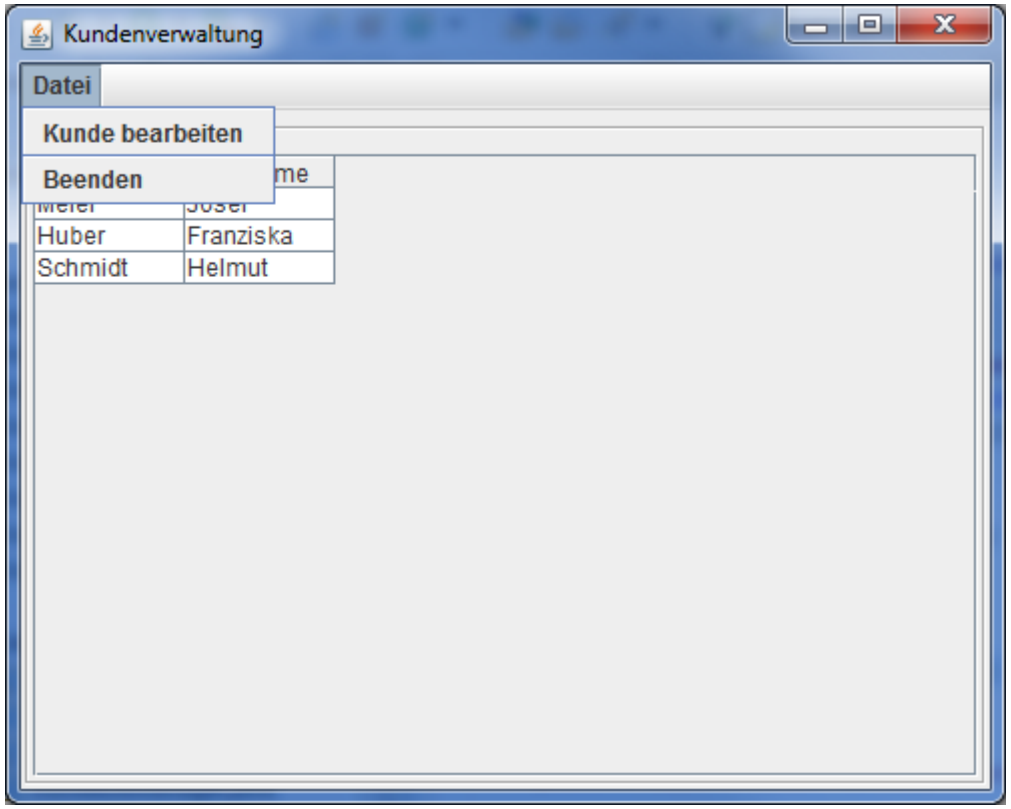
Anbinden des KundeFileManagers

```
public class KundenVerwaltung extends JFrame { …
   private KundeFileManager kundeFileManager;

   public KundenVerwaltung() { …
      menu.add(mi = new JMenuItem("Kundenstamm speichern"));
      mi.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
            try {
               kundeFileManager.save(kundeTableModel.getKunden());
            }
            catch (Exception ex) {}
         }
      }); …
   } …
}
```

KundenVerwaltung v8



Neue Features:

- Fast keine Änderung der GUI
- Aber: Nutzen einer Datenbank statt CSV-Dateien
- Laden der Daten zu Beginn, speichern der einzelnen Datensätze nach ihrer Änderung im Dialogfenster.

Datenbank Grundlagen

- **Datenbanken** sind semantisch zusammengehörige Daten, die von einem Datenbankmanagementsystem (DBMS) verwaltet werden.

Datenbank Grundlagen

- Ein **DBMS** kapselt die Verwaltung persistenter Daten von deren Nutzung durch ein Anwendungssystem. Semantisch zusammengehörige Daten werden in so genannten Datenbanken vereint.
- Die derzeit populärsten DBMS sind der Reihe nach Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, DB2 und Microsoft Access (Quelle: https://db-engines.com/de/ranking)

Datenbank Grundlagen

- Die gängigsten DBMS verwalten sog. **Relationale Datenbanken**.
- Relationale Datenbanken bestehen aus **Tabellen**. Einzelne Zeilen, sog. **Datensätze**, werden durch ihren **Primärschlüssel** identifiziert.
- Beziehungen zwischen Tabellen werden durch den **Fremdschlüssel** hergestellt. Dieser beinhaltet konkret den Primärschlüssel des referenzierten Datensatzes in dessen Tabelle.

Grundlagen SQL

- SQL (Structured Query Language) ist eine Abfrage- und Manipulationssprache für relationale Datenbanken.
- (Die SQL-Syntax auf den folgenden Folien ist nicht vollständig.)
- Grundsätzlich gilt: Zeichenketten müssen durch einfache Hochkomma eingeklammert werden.

Abfragen mittels SQL

- Syntax:
  SELECT *Spaltenauswahlliste*
  FROM *Tabellenliste*
  [WHERE *Bedingung*]

- Statt einer Spaltenauswahlliste kann auch ein Stern verwendet werden, um alle Spalten auszuwählen.
- Beispiele:
  - SELECT id, nachname, vorname FROM kunden
  - SELECT * FROM kunden

Einfügen mittels SQL

- Syntax:
  INSERT INTO *Tabellenname*
  [(*Spaltenliste*)]
  VALUES (*Werteliste*)

- Die Spaltenliste kann weggelassen werden, wenn für alle Spalten beim Einfügen Werte in der richtigen Reihenfolge angegeben werden.
- Beispiel:
  - INSERT INTO kunden (nachname, rabattsatz)
    VALUES ('Meier', 0.3)

Aktualisieren mittels SQL

- Syntax:
  UPDATE *Tabellenname*
  SET { *Spalte* = *Spaltenwert*} [,…]
  [WHERE *Bedingung*]

- Wird die WHERE-Bedingung weggelassen, so gilt der Update für alle Sätze!
- Beispiel:
  - UPDATE kunden
    SET nachname = ’Meier’, vorname = ’Josef’
    WHERE id = 1

Löschen mittels SQL

- Syntax:
  DELETE FROM *Tabellenname*
  [WHERE *Bedingung*]

- Wird die WHERE-Bedingung weggelassen, so werden alle Sätze der Tabelle gelöscht!
- Beispiel:
  - DELETE FROM kunden
    WHERE id = 2

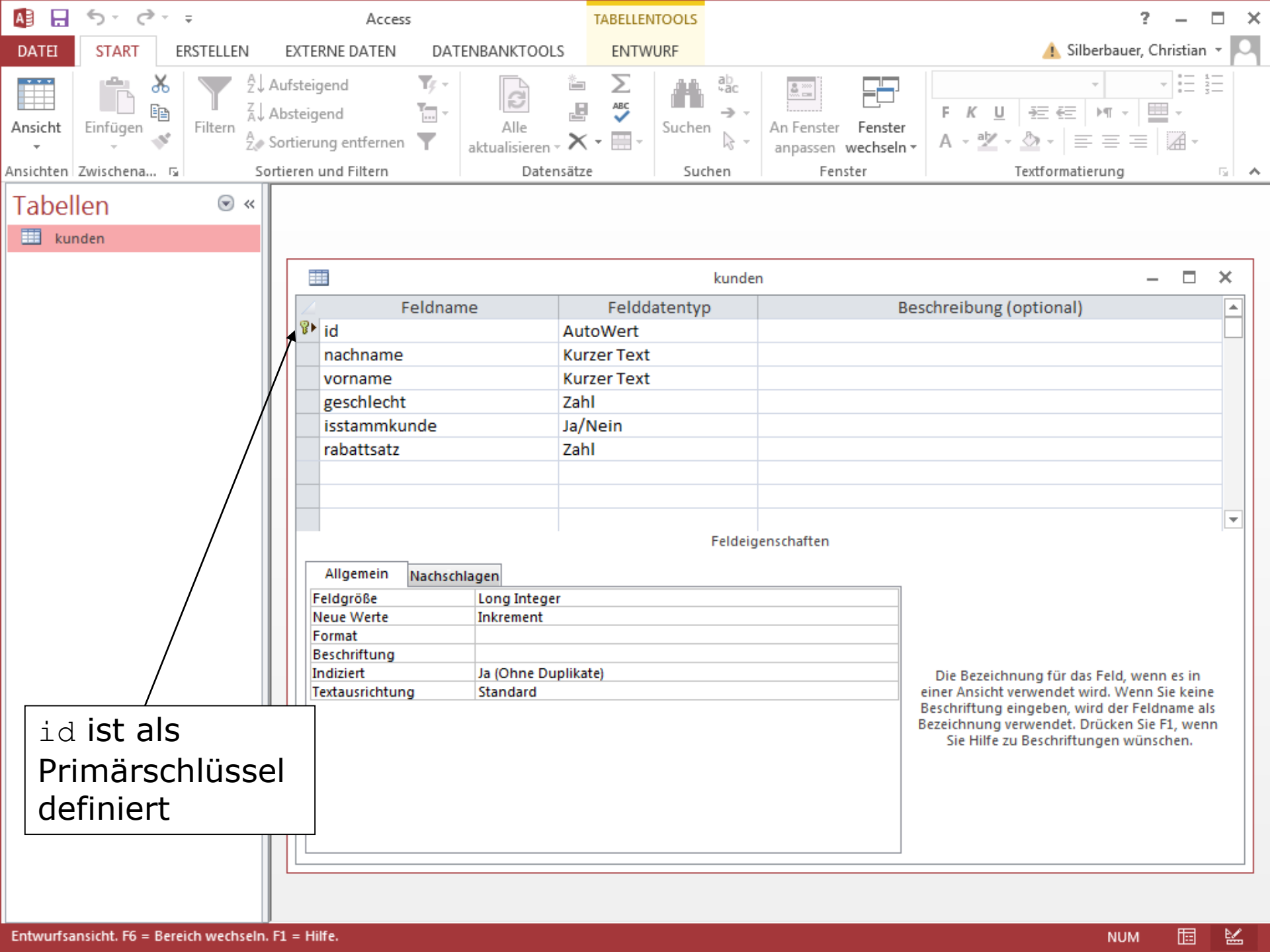
Vorteile gegenüber Dateien

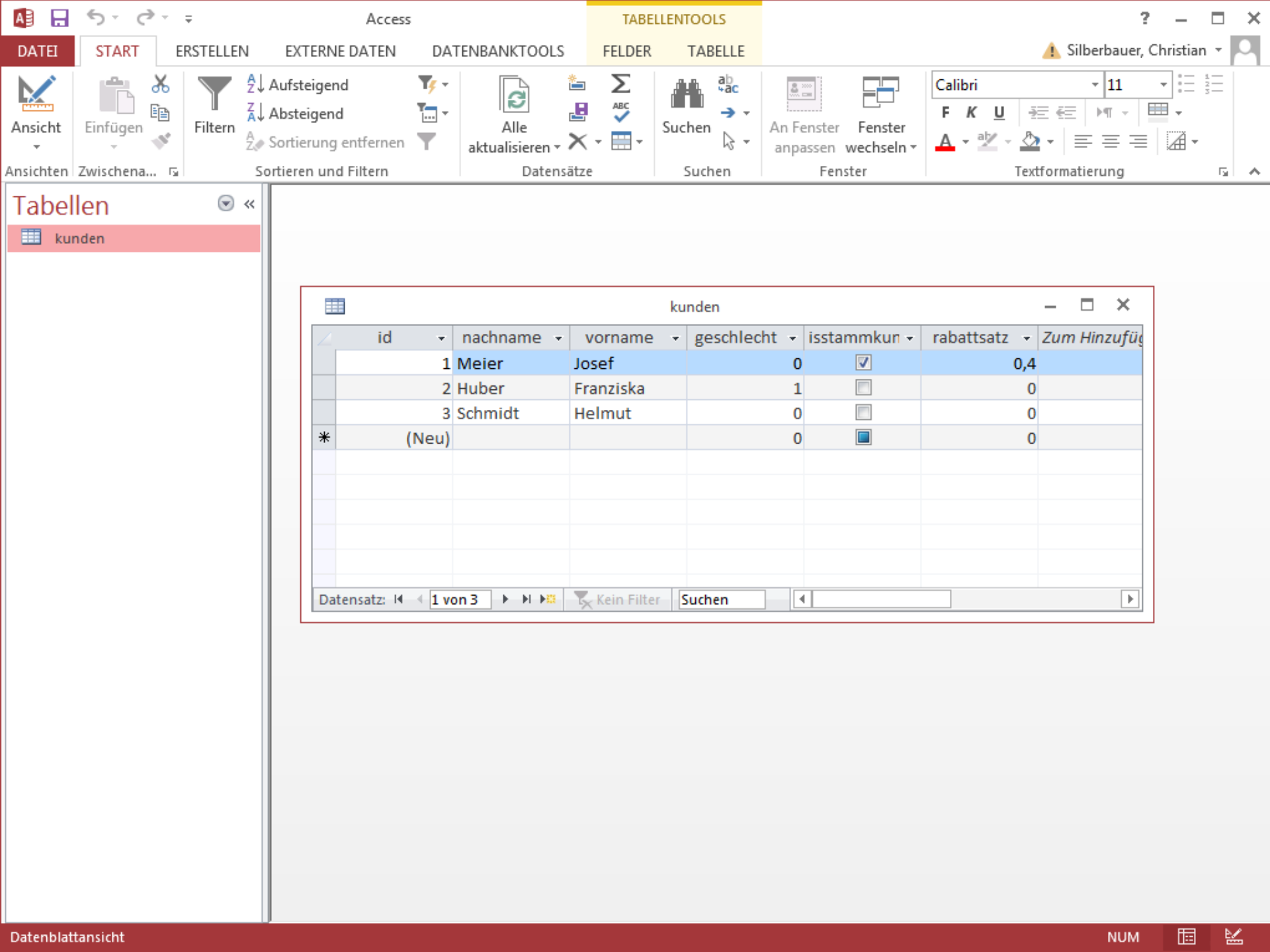
- Wir müssen uns nicht überlegen, *wie* die Daten gespeichert werden, d.h. in welchem Format.
- Abfragen und Manipulationen können mit Hilfe von SQL wesentlich einfacher (und i.d.R. auch performanter) erledigt werden, als durch einen direkten Dateizugriff.

- Weitere Details zum Thema *Datenbanken* siehe: http://www.javaundoop.de/extras/datenbanken.pdf

Datenbank vorbereiten

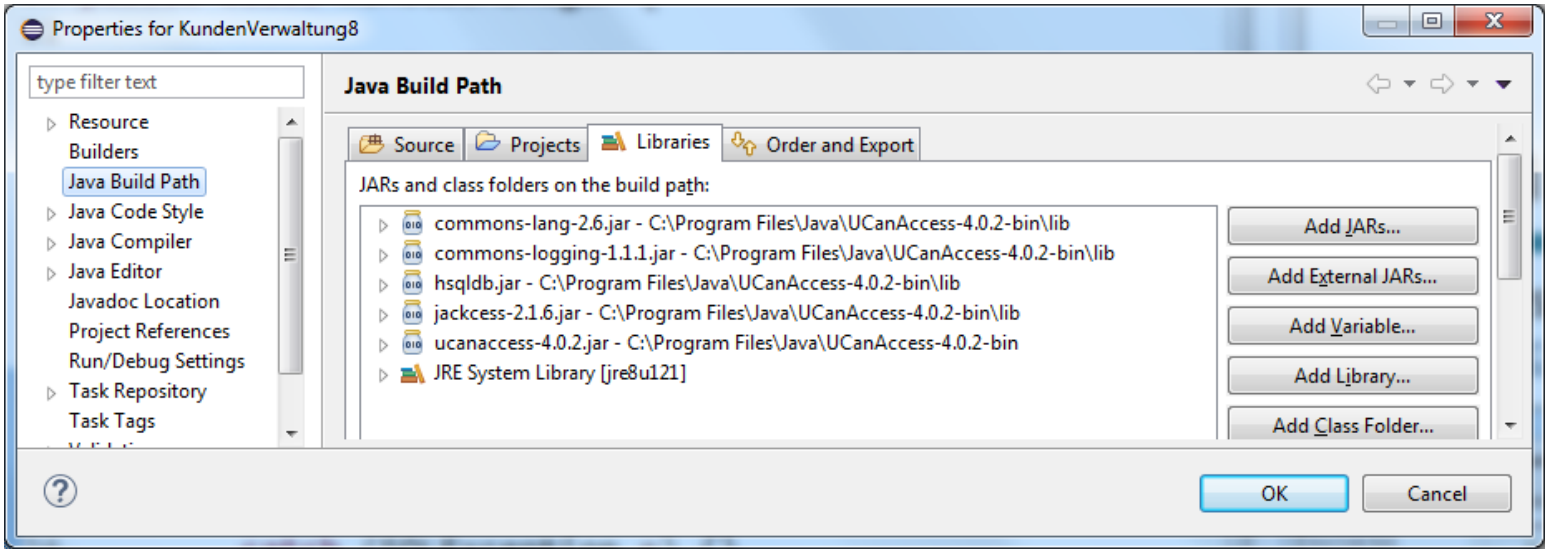
- Access-Datenbank ***kundenverwaltung.mdb*** anlegen
- Tabelle ***kunden*** erstellen
- JDBC-Treiber einrichten


Access
TABELLENTOOLS
DATEI
START
ERSTELLEN
EXTERNE DATEN
DATENBANKTOOLS
ENTWURF
Silberbauer, Christian
Ansicht
Einfügen
Filtern
Aufsteigend
Absteigend
Sortierung entfernen
Alle aktualisieren
Suchen
An Fenster anpassen
Fenster wechseln
Ansichten
Zwischena...
Sortieren und Filtern
Datensätze
Suchen
Fenster
Textformatierung
Tabellen
kunden
kunden
Feldname
Felddatentyp
Beschreibung (optional)
id
AutoWert
nachname
Kurzer Text
vorname
Kurzer Text
geschlecht
Zahl
isstammkunde
Ja/Nein
rabattsatz
Zahl
Feldeigenschaften
Allgemein
Nachschlagen
Feldgröße
Long Integer
Neue Werte
Inkrement
Format
Beschriftung
Indiziert
Ja (Ohne Duplikate)
Textausrichtung
Standard
Die Bezeichnung für das Feld, wenn es in einer Ansicht verwendet wird. Wenn Sie keine Beschriftung eingeben, wird der Feldname als Bezeichnung verwendet. Drücken Sie F1, wenn Sie Hilfe zu Beschriftungen wünschen.
id ist als Primärschlüssel definiert
Entwurfsansicht. F6 = Bereich wechseln. F1 = Hilfe.
NUM


Access
TABELLENTOOLS
DATEI
START
ERSTELLEN
EXTERNE DATEN
DATENBANKTOOLS
FELDER
TABELLE
Silberbauer, Christian
Ansicht
Einfügen
Filtern
Aufsteigend
Absteigend
Sortierung entfernen
Alle aktualisieren
Suchen
An Fenster anpassen
Fenster wechseln
Calibri
11
Ansichten
Zwischena...
Sortieren und Filtern
Datensätze
Suchen
Fenster
Textformatierung
Tabellen
kunden
kunden
id
nachname
vorname
geschlecht
isstammkun
rabattsatz
Zum Hinzufüg
1 Meier Josef 0 0,4
2 Huber Franziska 1 0
3 Schmidt Helmut 0 0
(Neu) 0 0
Datensatz:
1 von 3
Kein Filter
Suchen
Datenblattansicht
NUM

JDBC-Treiber einrichten

- Download der aktuellen Version des Treibers *UCanAccess* von http://ucanaccess.sourceforge.net/site.html
- Referenzieren der enthaltenen Bibliotheken im Projekt unter Properties/Java Build Path:



KundeDbManager

```
public class KundeDbManager {

   private static final String DBPATH = "db/kundenverwaltung.mdb";
   private Connection conn;

   public KundeDbManager() throws SQLException, ClassNotFoundException {
      conn = DriverManager.getConnection("jdbc:ucanaccess://" + DBPATH);
   }

   public void closeConnection() {
      try {
         conn.close();
      }
      catch (SQLException e) {}
   }
   …
}
```

KundeDbManager.load()

```
public class KundeDbManager { …
   public List<Kunde> load() throws SQLException {
      Statement st = conn.createStatement();
      String sqlcmd = "select * from kunden";
      ResultSet res = st.executeQuery(sqlcmd);
      List<Kunde> kunden = new ArrayList<Kunde>();

      while (res.next()) {
         Kunde kunde = new Kunde();
         kunde.setId(res.getInt("id"));
         kunde.setNachname(res.getString("nachname"));
         kunde.setVorname(res.getString("vorname"));
         kunde.setGeschlecht(Geschlecht.values()[res.getInt("geschlecht")]);
         kunde.setStammkunde(res.getBoolean("isstammkunde"));
         kunde.setRabattSatz(res.getDouble("rabattsatz"));
         kunden.add(kunde);
      }
      st.close();
      return kunden;
   } …
}
```

KundeDbManager.save()

```
public class KundeDbManager { …
   public void save(Kunde kunde) throws SQLException {
      String prepsqlcmd = "update kunden set nachname = ?, vorname = ?,"
                  + " geschlecht = ?, isstammkunde = ?, rabattsatz = ?"
                  + " where id = ?";
      PreparedStatement st = conn.prepareStatement(prepsqlcmd);

      st.setString(1, kunde.getNachname());
      st.setString(2, kunde.getVorname());
      st.setInt(3, kunde.getGeschlecht().ordinal());
      st.setBoolean(4, kunde.isStammkunde());
      st.setDouble(5, kunde.getRabattSatz());
      st.setInt(6, kunde.getId());

      st.executeUpdate();
      st.close();
   }
}
```

Anbinden des `KundeDbManager`s

```
public class KundenVerwaltung extends JFrame { …
   private KundeDbManager kundeDbManager;

   public KundenVerwaltung() throws Exception { …
      kundeDbManager = new KundeDbManager();
      kundeTableModel.setKunden(kundeDbManager.load());
      setVisible(true);
   }
   private void editKunde() {
      int row = kundeSelectionModel.getMinSelectionIndex();
      Kunde kunde = kundeTableModel.getKunde(row);
      new KundeBearbeitenDialog(this, kunde);
      kundeTableModel.fireTableRowsUpdated(row, row);
      try {
         kundeDbManager.save(kunde);
      }
      catch (Exception e) { }
   } …
}
```

Vor `setVisible()` kein `fireTable-DataChanged()` notwendig

JDBC

- Mit JDBC bietet die Java-Plattform eine einheitliche Möglichkeit um grundsätzlich auf beliebige tabellarische Daten zuzugreifen, meist aber wird sie für den Zugriff auf Relationale Datenbanken verwendet.
- UCanAccess ist ein JDBC-Treiber für Microsoft Access.

Aufbau einer Verbindung

- Mit Hilfe der Klasse `DriverManager` kann eine Verbindung zum JDBC-Treiber und damit zur Datenbank hergestellt werden.
- Eine Verbindung wird durch den Typ `Connection` abgebildet.
- Alternativ zu `DriverManager` – laut Oracle die empfohlene Lösung – kann ein `DataSource`-Objekt verwendet werden.

Datenbankverbindungen

- Per `Connection` können SQL-Anweisungen vom Typ `Statement` erzeugt werden (`createStatement()`).
- Alternativ kann mit `prepareStatement()` eine „vorbereitete“ SQL-Anweisung genutzt werden.
- Zum Schließen einer `Connection` kann die Methode `close()` verwendet werden, um Ressourcen freizugeben.

SQL-Anweisungen

- SQL-Anweisungen werden durch den Typ `Statement` repräsentiert.
- Für SQL-Abfragen kann die Methode `executeQuery()` genutzt werden, welche eine Ergebnismenge bzw. ein `ResultSet` zurückgibt.
- Für Datenmanipulationen in Tabellen steht die Methode `executeUpdate()` zur Verfügung. Sie liefert als Ergebnis die Anzahl der geänderten Datensätze.
- Zum Schließen einer Anweisung kann die Methode `close()` verwendet werden.

„Vorbereitete“ SQL-Anweisungen

- Statt einem gewöhnlichen `Statement` kann auch ein `PreparedStatement` verwendet werden.
- Während bei einem gewöhnlichen `Statement` die SQL-Anweisung direkt an das DBMS gesendet wird, wird sie mit einem `PreparedStatement` vorübersetzt.
- Muss das gleiche `Statement` viele Male ausgeführt werden, verbessert ein `PreparedStatement` daher normalerweise die Performance, sofern diese vom DBMS unterstützt werden.
- Ein `PreparedStatement` kann parametrisiert sein. Parameter werden durch ein „?“ gekennzeichnet.
- KV8 bietet keine gute Implementierung eines `PreparedStatement`, da dieses bei jedem Speichervorgang neu erzeugt wird.

Verarbeiten von SQL-Abfragen

- Eine SQL-Abfrage gibt als Ergebnis i.d.R. ein `ResultSet` zurück.
- Über dieses können die Ergebniswerte ausgelesen werden.
- Die Feldwerte des markierten Datensatzes, bzw. auf den der *Cursor* gesetzt ist, können mit typspezifischen get-Methoden ausgelesen werden.
- Die Methode `next()` setzt den Cursor auf den nächsten Satz und gibt `true` zurück, falls der letzte Satz noch nicht erreicht war. Initial ist der *Cursor* vor dem ersten Datensatz gesetzt.
- Auch ein `ResultSet` kann mit `close()` geschlossen werden, um Ressourcen freizugeben. Mit dem Schließen des Statements geschieht dies implizit mit.