Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

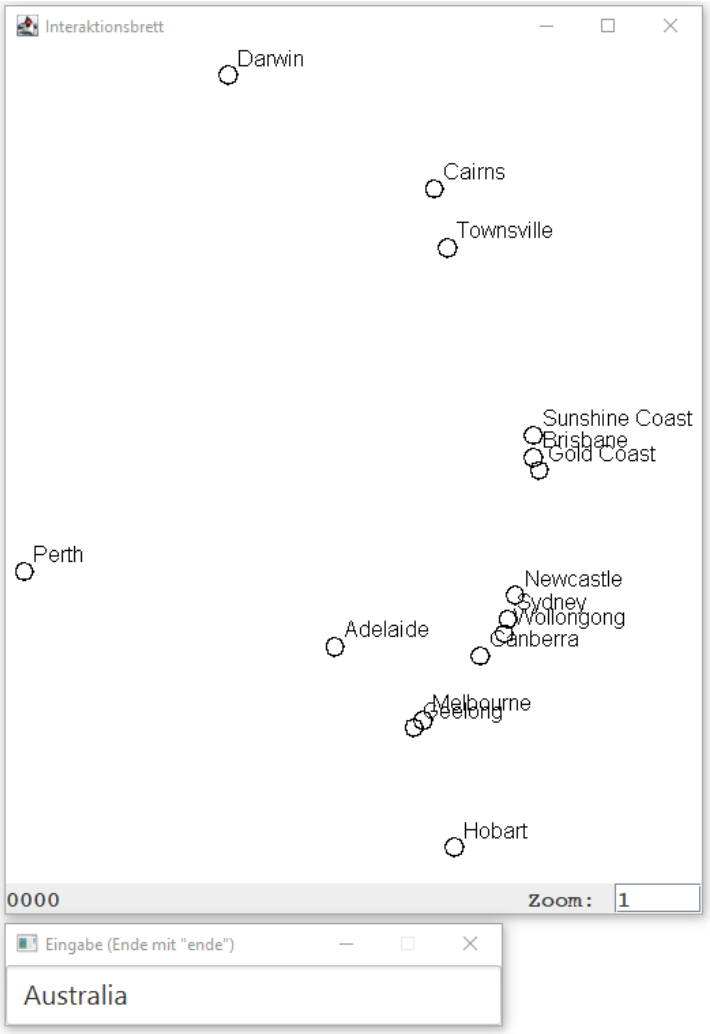
**Datenbanksysteme**

Wintersemester 2023/24
**10. Aufgabenblatt**

## Aufgabe 25 (3 Punkte)

Schreiben Sie ein Java-Programm mit dem alle in Mondial mit Längen- und Breitengrad eingetragenen Städte eines Landes in einer Karte eingezeichnet werden. In dem von der Veranstaltungsseite erhältlichen Projekt ist bereits eine Eingabezeile realisiert, in die der Name eines Landes eintragen und Return gedrückt werden kann. Ihre Aufgabe besteht darin, eine Klasse zu schreiben, die das Interface karte.Karte realisiert. Achten Sie darauf,

- dass alle Städte angezeigt werden mit ihrer Position,
- dass das Programm bei nicht existierenden Ländern oder Ländern bei denen die Gradangaben bei allen Städten fehlen, nicht abstürzt,
- dass es eine sinnvolle Ausgabe gibt, wenn nur eine Stadt eingetragen ist.
- dass der Platz zur Ausgabe ausgenutzt wird, also Länder so vollständig den Platz der Anzeige nutzen (auf Breite und Höhe verteilen).




Zum Testen eigenen sich u.a. Andorra und Germany. Die rechte Seite zeigt eine Beispielausgabe für Australia. Ausgaben dürfen sich überlappen.
Die Karte wird mit Hilfe der Klasse Interaktionsbrett angezeigt, über die man sich auf der Seite http://kleuker.iui.hs-osnabrueck.de/querschnittlich/Interaktionsbrett/index.html informieren kann.

## Aufgabe 26 (5 Punkte)

a) Legen Sie eine Tabelle der folgenden Form in einer neuen Datenbank an.

```
CREATE TABLE Studierend(
  Matnr INTEGER,
  Name VARCHAR(16),
  Semester VARCHAR(6),
  CONSTRAINT PK_Student
     PRIMARY KEY(matnr)
);
```

b) Schreiben Sie basierend auf dem Teilprojekt von der Veranstaltungswebseite ein Java-Programm in der Klasse `dialog.Studierendenbearbeitung`, mit dem man
   - das Programm beenden
   - Studierende ergänzen (scheitert, wenn Matrikelnummer vorhanden)
   - alle Studierenden ausgeben

- die Namen der Studierenden ändern
- alle Studierenden löschen

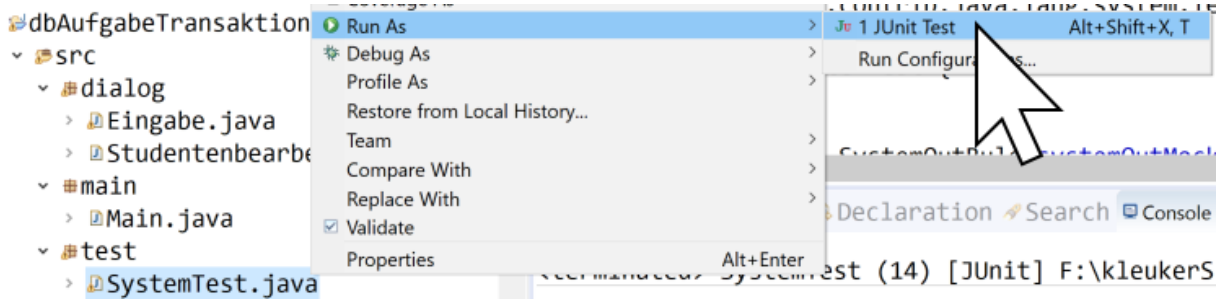
kann. Der folgende Nutzungsdialog soll z. B. möglich sein, Eingaben sind umrandet. Das System soll melden, ob die Änderung durchgeführt wurde oder nicht.

```
Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
2
42: Ute (WiSe19)
43: Uwe (WiSe19)
44: Urs (SoSe20)

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
4

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
2

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
1
Name: Marion
Matrikelnummer: 72
Semester: WiSe19
erfolgreich hinzugefuegt

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
2
72: Marion (WiSe19)

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
3
Matrikelnummer: 73
neuer Name: Hans
nicht geaendert

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
3
Matrikelnummer: 72
neuer Name: Susanne
erfolgreich geaendert

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
2
72: Susanne (WiSe19)

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
1
Name: Ernst
Matrikelnummer: 72
Semester: WiSe20
nicht hinzugefuegt

Was wollen Sie?
(0) Programm beenden
(1) neuen Studi hinzufuegen
(2) alle Studis zeigen
(3) Namen eines Studi aendern
(4) alle Studis loeschen
0
```

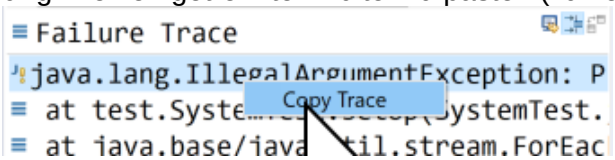
Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

**Datenbanksysteme**

Wintersemester 2021/22
**10. Aufgabenblatt**




Da es hier um die Funktionsweise geht, können Sie auf ergonomische Prüfungen von Fehleingaben verzichten und bei einer SQLException diese einfach fangen, das Programm soll aber nicht abstürzen. Erzeugen Sie *genau* die im Beispieldialog gezeigten Ausgaben, z. B. Matrikelnr:␣Name␣(Semester), da es sonst Probleme mit den Tests geben kann.
Nutzen Sie zum Testen des Programms die Klasse SystemTest (die unabhängig von den folgenden Teilaufgaben nutzbar sein sollte). Nutzen Sie zunächst die Standardtransaktionseinstellungen von Java (Erinnerung: autoCommit ist true).
Hinweis: Um sich JUnit-Meldungen genauer anzusehen, ist es sinnvoll, einen Rechtsklick auf dem Failure Trace zu machen, „Copy Trace“ auszuwählen und die genaue Fehlermeldung in einen getrennten Editor zu pasten (zu kopieren, Strg+V).




c) Führen Sie Ihr Programm gleichzeitig mehrfach in verschiedenen Eclipse-Fenstern aus und notieren Sie, was passiert, wenn Sie sich bei der ersten Nutzung und dann bei der zweiten Nutzung sich alle Studierenden zuerst anschauen (Menüpunkt 2) und danach fast gleichzeitig den Namen einer studierenden Person ändern (erster Nutzung beginnt Änderung, zweite Nutzung beginnt Änderung, erste Nutzung schließt Änderung ab, zweite Nutzung schließt Änderung ab)?

d) Ändern Sie Ihr Programm so ab, dass Sie für das Connection-Objekt `con` unmittelbar nach seiner Erzeugung die Methode `this.con.setAutocommit(false)` und erst bei der Beendigung des Programms die Methode `this.con.commit()` aufrufen. Damit finden hier nur zum Experimentieren und für die Praxis sehr ungewöhnlich alle Schritte im Programm in einer einzigen Transaktion statt. Führen Sie die Analyse aus c) für folgende Situationen durch, wenn unmittelbar nach `this.con.setAutocommit(false)` jeweils folgende Zeile ergänzt wird. Beachten Sie, dass vor einem neuen Experiment alle Programme wieder geschlossen sind und das Programm neu aufgerufen wird.
   - `this.con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);`
   - `this.con.setTransactionIsolation(Connection.TRANSACTION_REPEATABLE_READ);`
   - `this.con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);`

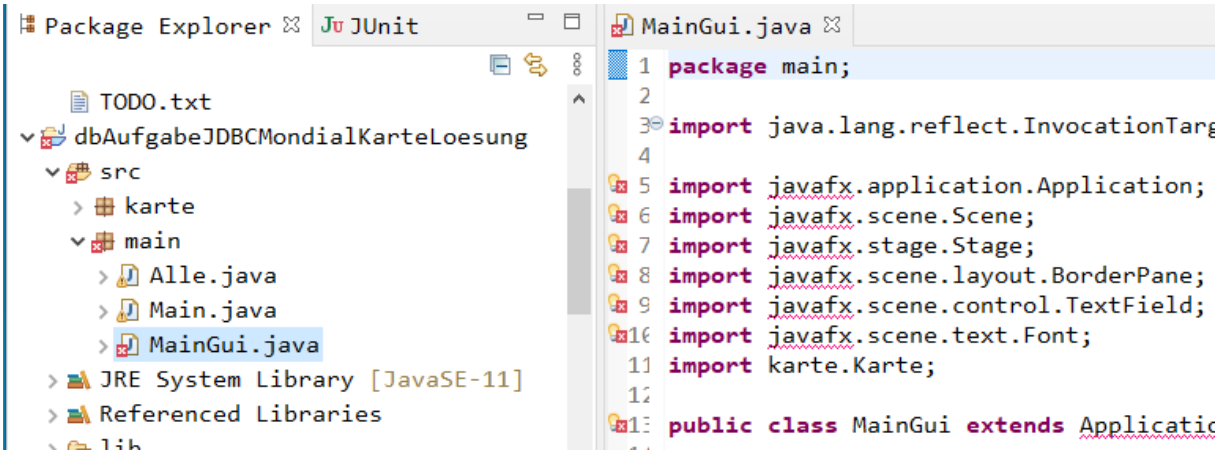
   Dokumentieren Sie jeweils Ihre Beobachtungen. Falls es so aussieht, dass das System nicht reagiert (warum?), warten Sie jeweils eine Minute.

mögliche Problembehebung:
Bei der Nutzung der KleukerSEU ist die Oberflächenbibliothek JavaFX (OpenJFX) bereits mit eingebunden. Gegebenenfalls kann es trotzdem passieren, dass sie nicht gefunden wird, was zu folgenden Fehlermeldungen in MainGui führt.

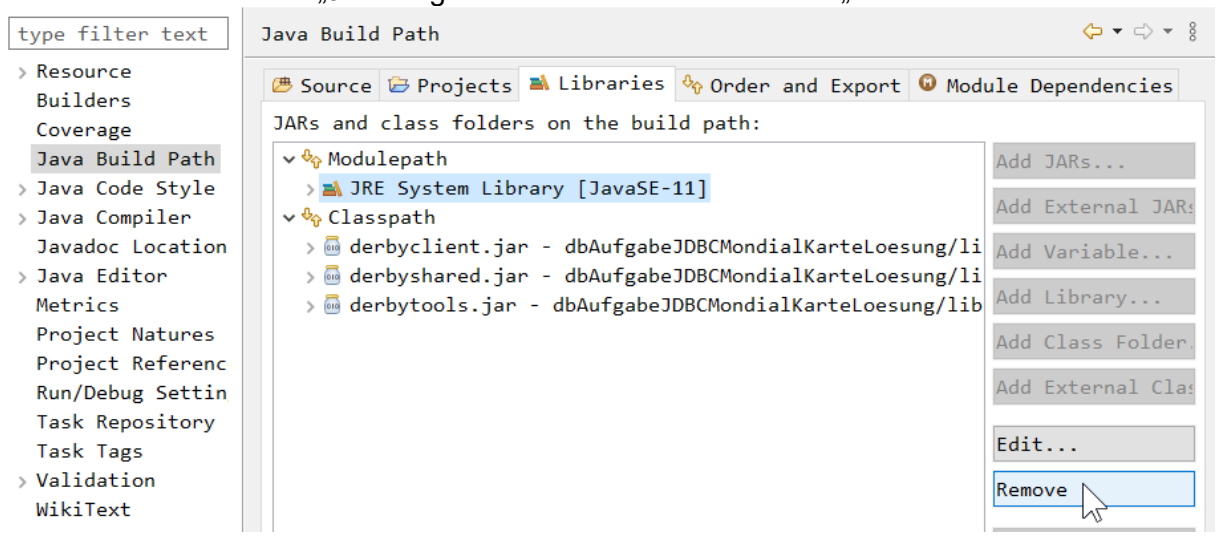
Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

**Datenbanksysteme**

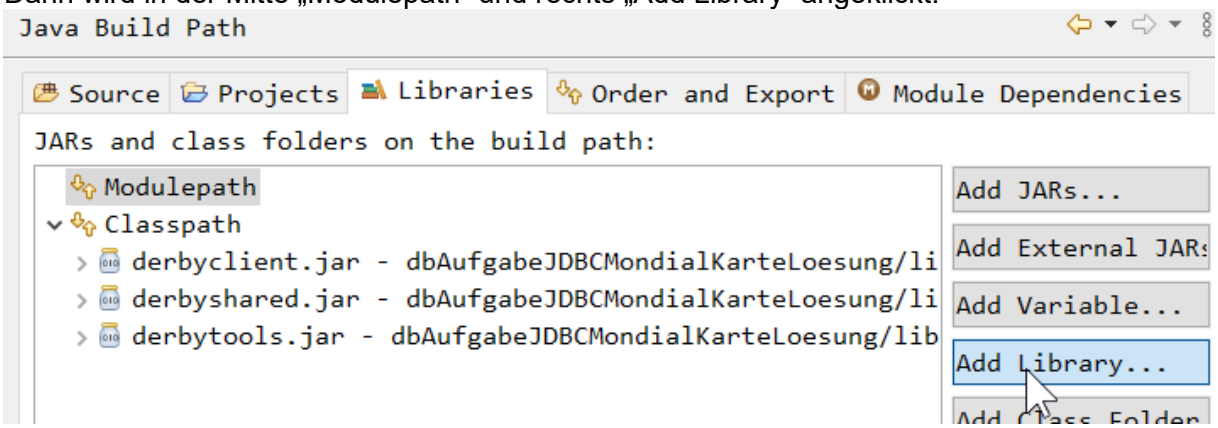
Wintersemester 2021/22
**10. Aufgabenblatt**



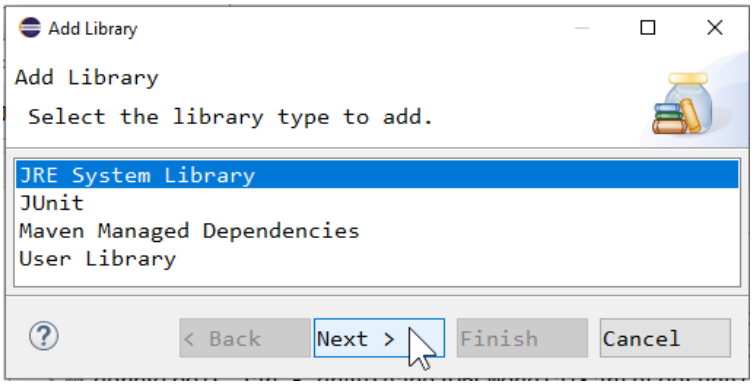

Dies deutet an, dass eine andere Java-Version als erwartet gerade genutzt wird. Diese Version muss jetzt neu eingestellt werden. Dazu wird nach einem Rechtsklick auf dem Projekt unten „Properties“ ausgewählt, rechts auf „Java Build Path“ und oben auf „Libraries“ geklickt. Zunächst wird dann auf „JRE…“ geklickt und dieses rechts über „Remove“ entfernt.



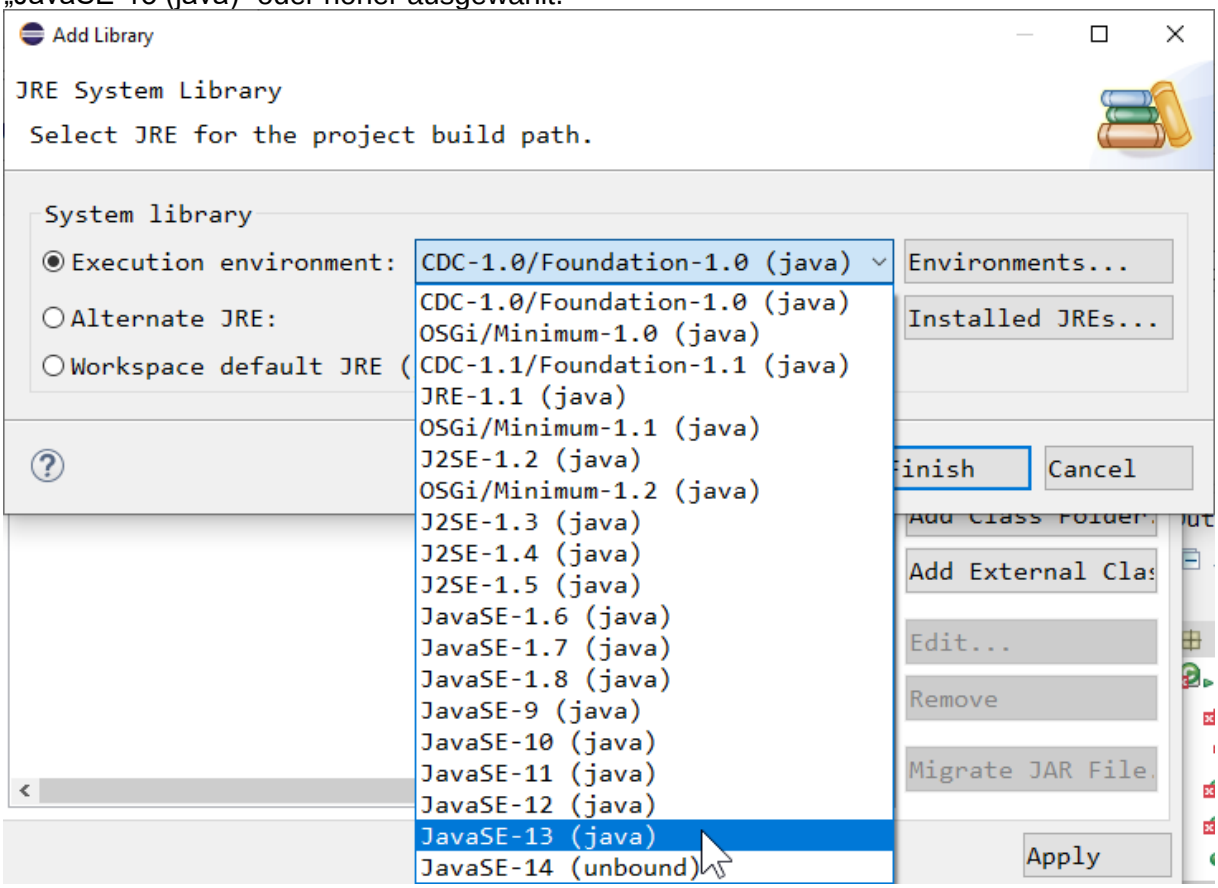

Dann wird in der Mitte „Modulepath“ und rechts „Add Library“ angeklickt.




Es wird oben „JRE System Library“ ausgewählt und „Next >“ geklickt.




Es wird oben „Execution environment“ und dann in der Drop-Down-Box rechts daneben „JavaSE-13 (java)“ oder höher ausgewählt.




Die Aktion wird mit „Finish“ und „Apply and Close“ abgeschlossen. Danach sollten alle Fehlermeldungen verschwunden sein.
Sollte es weiterhin Probleme in der KleukerSEU geben, melden Sie sich bei Prof. Dr. Kleuker (E-Mail, Vorlesungssprechstunde).

zu A25)
Anfrage zur Suche von Ländern mit nur einer Stadt:

```
SELECT Country.Name
  FROM Country, City
  WHERE Country.Code = City.Country
  GROUP BY Country.Name
  HAVING COUNT(*) = 1;
```

```java
package karte;

public interface Karte {
    /**
     * Zum uebergebenen Land werden alle
     * zugehorigen Staedte aus Mondial
     * ausgegeben
     * @param country betrachtetes Land
     */
    public void zeigeLand(String country);

    /** Methode dient zum Beenden des
     * Programms und muss nur die
     * Datenbankverbindung trennen.
     */
    public void beenden();
}
```

```java
package karte;

public class Stadt {
    private String name;
    private Double longitude;
    private Double latitude;

    public Stadt(String name, Double longitude, Double latitude) {
        this.name = name;
        this.longitude = longitude;
        this.latitude = latitude;
    }

    public String getName() {
        return name;
    }

    public Double getLongitude() {
        return longitude;
    }

    public Double getLatitude() {
        return latitude;
    }

    @Override
    public String toString() {
        return "Stadt{" + "name=" + name + ", longitude="
               + longitude + ", latitude=" + latitude + '}';
```

```
    }
}

package karte;

import java.lang.reflect.InvocationTargetException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class Kartenrealisierung implements Karte {

  private static Interaktionsbrett ia;
  private Connection con;
  private Statement stmt;
  private List<Stadt> staedte;
  private double minlong = 180;
  private double maxlong = -180;
  private double minlat = 180;
     private double maxlat = -180;

  public Kartenrealisierung() {
    if (ia == null) {
      ia = new Interaktionsbrett();
    }
    try {
      Class.forName("org.apache.derby.jdbc.ClientDriver")
          .getDeclaredConstructor()
          .newInstance();

      // mein DB-Ordner F:\workspaces\datenbanken\Mondial
      this.con = DriverManager.getConnection(
          "jdbc:derby://localhost:1527/"
          + "F:\\workspaces\\datenbanken\\Mondial"
              , "kleuker"
              , "kleuker");
      this.stmt = con.createStatement();
      System.out.println("Datenbank verbunden");
    } catch (SQLException | ClassNotFoundException |
InstantiationException
        | IllegalAccessException
        | IllegalArgumentException | InvocationTargetException
        | NoSuchMethodException | SecurityException ex) {
      System.out.println("Problem: " + ex);
```

```
  }
}

@Override
public void zeigeLand(String country) {
  this.staedte = new ArrayList<>();
  System.out.println("stmt: " +stmt);
  try (ResultSet rs = stmt.executeQuery(
          "SELECT City.Name,Longitude,Latitude "
          + "    FROM City, Country "
            + "    WHERE City.Country = Country.Code "
          + "      AND Country.Name = '" + country + "'")) {
    while (rs.next()) {
      String name = rs.getString(1);
      double longitude = rs.getDouble(2);
      if (!rs.wasNull()) {
        double latitude = rs.getDouble(3);
        if (!rs.wasNull()) {
          this.staedte.add(new Stadt(name, longitude, latitude));
        }
      }
    }
  } catch (SQLException ex) {
    System.out.println("Problem: " + ex);
  }
  ;
  if (this.staedte.isEmpty()) {
    System.out.println("keine Staedte mit verwertbaren Daten");
  } else {
    ia.zuruecksetzen();
    System.out.println(this.staedte);
    if (this.staedte.size() == 1) {
      ia.neuerKreis(150, 200, 5);
      ia.neuerText(160, 200, this.staedte.get(0).getName());
    } else {
      this.grenzen();
      this.zeichnen();
    }
  }
}

private void zeichnen() {
  this.staedte.forEach(s -> {
    int x = 5 + (int) (280 * (s.getLongitude()
        - this.minlong) / (this.maxlong - this.minlong));
    int y = 430 - (int) (420 * (s.getLatitude()
        - this.minlat) / (this.maxlat - this.minlat));
    ia.neuerKreis(x, y, 5);
```

```
      ia.neuerText(x + 10, y, s.getName());
    });
  }

  private void grenzen() {
    this.minlong = 180;
    this.maxlong = -180;
    this.minlat = 180;
    this.maxlat = -180;
    this.staedte.forEach(s -> {
      if (s.getLongitude() < this.minlong) {
        this.minlong = s.getLongitude();
      }
      if (s.getLongitude() > this.maxlong) {
        this.maxlong = s.getLongitude();
      }
      if (s.getLatitude() < this.minlat) {
        this.minlat = s.getLatitude();
      }
      if (s.getLatitude() > this.maxlat) {
        this.maxlat = s.getLatitude();
      }
    });
  }

  @Override
  public void beenden() {
    try {
      if (!this.stmt.isClosed()) {
        this.stmt.close();
      }
      if (!this.con.isClosed()) {
        this.con.close();

      }
    } catch (SQLException ex) {
      System.out.println("Problem: " + ex);
    }
  }
}
```

---

zu A26)

Studis haben Begriff der Transaktion zwar gehört, ist aber noch nicht systematisch in der VL behandelt worden; dies wird später in der VL nachgeholt.

Beachten: Tests suchen exakt nach der angegebenen Form der Ausgabe. Falls Tests scheitern und man meint, das Programm läuft, dann Testdaten von Hand prüfen.

Natürlich wäre die Nutzung von PreparedStatements etwas schöner.

```
package dialog;

import java.lang.reflect.InvocationTargetException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Studentenbearbeitung {

  private Connection con;
  private Statement stmt;

  /**
   * Baut eine Verbindung zur vorher eingerichteten Datenbank auf. Den
Variablen
   * con und stmt werden passende Objekte zugeordnet.
   *
   * @return Ergebnis gibt an, ob der Verbindungsaufbau erfolgreich war
   */
  public boolean verbinden() {
    try {
      Class.forName("org.apache.derby.jdbc.ClientDriver")
          .getDeclaredConstructor()
          .newInstance();

      this.con = DriverManager
          .getConnection("jdbc:derby://localhost:1527/"
              + "Z:\\tmp\\StudentTransaktionssteuerung;create=true",
              "kleuker", "kleuker");
      this.stmt = con.createStatement();
      System.out.println("Datenbank verbunden");
      return true;
    } catch (SQLException | ClassNotFoundException | InstantiationException
        | IllegalAccessException | IllegalArgumentException
        | InvocationTargetException | NoSuchMethodException
        | SecurityException ex) {

Logger.getLogger(Studentenbearbeitung.class.getName()).log(Level.SEVERE,
          null, ex);
    }
    return false;
  }

  /**
   * Schliesst die Datenbankverbindung.
   */
  public void schliessen() {
    try {
```

```
      if (!this.stmt.isClosed()) {
        this.stmt.close();
      }
      if (!this.con.isClosed()) {
        this.con.close();

      }
    } catch (SQLException ex) {
      Logger.getLogger(Studentenbearbeitung.class
          .getName()).log(Level.SEVERE, null, ex);
    }
  }

  public void starten() {
    if (this.verbinden()) {
      try {
        this.con.setAutoCommit(false);
/* einen der naechsten 4 jeweils auswaehlen
this.con.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED);
this.con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
this.con.setTransactionIsolation(Connection.TRANSACTION_REPEATABLE_READ);
*/
this.con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

        this.dialog();
        this.con.commit();
        this.schliessen();
      } catch (SQLException ex) {

Logger.getLogger(Studentenbearbeitung.class.getName()).log(Level.SEVERE,
            null, ex);
      }
    }
  }

  public void dialog() {
    int eingabe = -1;
    while (eingabe != 0) {
      System.out.println("\nWas wollen Sie?\n"
          + "(0) Programm beenden\n"
          + "(1) neuen Studi hinzufuegen\n"
          + "(2) alle Studis zeigen\n"
          + "(3) Namen eines Studi aendern\n"
          + "(4) alle Studis loeschen");
      eingabe = Eingabe.leseInt();
      switch (eingabe) {
      case 1:
        this.studiHinzu();
        break;
      case 2:
        this.alleStudis();
        break;
```

```
      case 3:
        this.studinamenAendern();
        break;
      case 4:
        this.studisLoeschen();
        break;
      }
    }
  }

  public void studinamenAendern() {
    System.out.print("Matrikelnummer: ");
    int mat = Eingabe.leseInt();
    System.out.print("neuer Name: ");
    if (this.studiAendern(mat, Eingabe.leseString())) {
      System.out.println("erfolgreich geaendert");
    } else {
      System.out.println("nicht geaendert");
    }
  }

  public void studiHinzu() {
    System.out.print("Name: ");
    String name = Eingabe.leseString();
    System.out.print("Matrikelnummer: ");
    int matnr = Eingabe.leseInt();
    System.out.print("Semester: ");
    String semester = Eingabe.leseString();
    if (this.studiHinzu(matnr, name, semester)) {
      System.out.println("erfolgreich hinzugefuegt");
    } else {
      System.out.println("nicht hinzugefuegt");
    }
  }

  // ab hier zu berarbeiten

  public void studisLoeschen() {
    try {
//            this.stmt.execute("DELETE FROM Hoert");
      this.stmt.execute("DELETE FROM Student");
    } catch (SQLException ex) {
      Logger.getLogger(Studentenbearbeitung.class.getName())
          .log(Level.SEVERE, null, ex);
    }
  }

  public void alleStudis() {
    try {
      ResultSet rs = this.stmt.executeQuery(
              "SELECT * FROM Student");
      while (rs.next()) {
```

```
        System.out.println(rs.getInt(1) + ": " + rs.getString(2)
            + " (" + rs.getString(3) + ")");
      }
    } catch (SQLException ex) {
      Logger.getLogger(Studentenbearbeitung.class.getName())
      .log(Level.SEVERE, null, ex);
    }

  }

  private boolean studiHinzu(int matnr, String name, String semester) {
    try {
      int anzahl = this.stmt.executeUpdate(
          "INSERT INTO Student VALUES("
              + matnr + ",'" + name + "','" + semester + "')");
      if (anzahl == 1) {
        return true;
      }
    } catch (SQLException ex) {
    }
    return false;
  }

  private boolean studiAendern(int matnr, String neu) {
    try {
      int anzahl = this.stmt.executeUpdate(
              "UPDATE Student SET name='" + neu
              + "' WHERE matnr=" + matnr);
      if (anzahl == 1) {
        return true;
      }
    } catch (SQLException ex) {
    }
    return false;
  }

}
```

zu c) Letzte Änderung gewinnt

zu d) Verhalten könnte im Detail abweichen, falls die Implementierung abweicht oder die Schritte nicht wie angegeben durchgeführt wurden. Jeder Studi sollte zumindest mal eine Sperre “mitbekommen” haben.

i) beim zweiten Nutzer wird die Übernahme der Änderung verweigert, genauer gewartet, bis der erste das Programm beendet, bei zu langem Warten bricht die Aktion ab.

ii) der erste Nutzer kann Änderung nicht abschließen, wenn er es versucht, beim zweiten Nutzer wird der Änderungsversuch sofort abgebrochen und die Änderung beim ersten Nutzer vollendet

iii) der erste Nutzer kann Änderung nicht abschließen, wenn er es versucht, gleiches gilt für den zweiten Nutzer, nach kurzer Zeit wird der zweite Änderungsversuch verworfen und der erste durchgeführt. In iii) wird damit ein Deadlock von der Datenbank aufgelöst. Versucht der zweite Nutzer danach sich alle Studis anzusehen, ist dies weiterhin nicht möglich und wird nach Ablauf einer Zeit abgebrochen.