

# Übung 05

Arbeitsaufwand insgesamt: 11h

## Inhaltsverzeichnis

Teil 1 – Zugriff auf eine Adress-Datenbank mit JDBC .....	3
Lösungsidee: .....	3
UML .....	4
Source-Code .....	5
PersonApplication.java .....	5
PersonController.java .....	5
PersonModel.java .....	7
PersonLogicImpl.java .....	8
PersonLogic.java .....	8
PersonTableModel.java .....	9
BindableView.java .....	10
PersonOverView.java .....	10
Testfälle .....	17
Nichtstarten der Datenbank .....	17
Leere Input-Felder .....	18
Falsches Input-Format .....	18
Update ohne Auswahl eines Eintrags .....	19
Löschen ohne Auswahl eines Eintrags .....	19
Teil 2 – Grafische Benutzerschnittstelle für Alignments .....	20
Lösungsidee .....	20
UML .....	21
Source-Code .....	22
AlignmentApplication.java .....	22
AlignmentItem.java .....	22
BindableView.java .....	24
AlignmentOverViewFrame.java .....	24
AlignmentListModel.java .....	33
AlignmentModel.java .....	33
AlignmentController.java .....	34

AlignmentLogic.java .....	36
AlignmentLogicImpl.java .....	36
Testfälle .....	37
Normale Benützung.....	37
Leere Sequenzen .....	37
Invalide Sequenzen.....	37
Remove ohne Selection.....	38
Sehr lange Sequenzen .....	38

## Teil 1 – Zugriff auf eine Adress-Datenbank mit JDBC

In diesem Teil der Übung soll die Datenbankschnittstelle vom letzten Übungszettel verwendet werden und auf dessen Basis ein GUI zur Verwaltung von Datenbankeinträgen für diese Datenbank erstellt werden.

### Lösungsidee:

Die Datenbankschnittstelle ist wie bereits erwähnt schon fertig. Das Ziel ist es nun ein UI zu entwickeln das mit diesem interagiert.

Auf Grundlage der Übungseinheiten kann dies leicht durch die Verwendung des MVC Patterns verwirklicht werden.

Der View enthält das „sichtbare UI“, alle Buttons und Eingabefelder zum Interagieren, sowie Elemente zum Anzeigen der Daten. In diesem Fall ist das eine Tabelle mit den Personendaten, sowie die Buttons zum Hinzufügen, Updaten und Entfernen von Personen.

Im Controller werden die, über die Buttons abgesetzten, Befehle über ActionListeners abgefangen und verarbeitet bzw. weiterdelegiert.

Im Model sind dann schlussendlich die Produktivdaten gespeichert: In unserem Fall wird das der Datentyp „Person“ werden, der in Collections gespeichert werden kann. Das Model enthält immer eine Momentaufnahme der gespeicherten Personen, sowie eine Person, über die der Controller, View und die Logik arbeiten.

Die Logik kümmert sich um die Verknüpfung zwischen dem Controller und der Datenbankschnittstelle und liefert booleans (beim Speichern und Löschen, als Erfolgsstatus) und Models (gesamte Daten, nach dem Updaten und Lesen von Personen).

Diese Anwendung wird dann über das Anlegen eines Controllers in der main Methode gestartet.

\*Im Endeffekt muss nur die EmployeeApplication aus der Übungseinheit umgeschrieben werden, um Personen zu verwalten, sowie die Funktionalitäten von „update“ und „remove“ hinzugefügt werden.

```

classDiagram
    class BindableView {
        +void bindModel(PersonModel)
        +PersonModel fillModel(PersonModel)
    }
    class PersonOverViewFrame {
        +void bindModel(PersonModel)
        +void clearFields()
        +void closeAddPersonDialog()
        +void closeRemovePersonDialog()
        +void closeUpdatePersonDialog()
        +PersonModel fillModel(PersonModel)
        +void initLayout()
        +void showAddPersonDialog()
        +void showError(String)
        +void showRemovePersonDialog()
        +void showUpdatePersonDialog()
        +PersonAddListener personAddListener
        +PersonRemoveListener personRemoveListener
        +PersonUpdateListener personUpdateListener
    }
    class PersonTableModel {
        +String getColumnName(int)
        +Object getValueAt(int, int)
        +int columnCount
        +int rowCount
    }
    class PersonLogic {
        +boolean deletePerson(PersonModel)
        +PersonModel readAllPersons(PersonModel)
        +boolean savePerson(PersonModel)
        +PersonModel updatePerson(PersonModel)
    }
    class PersonLogicImpl {
        +boolean deletePerson(PersonModel)
        +PersonModel readAllPersons(PersonModel)
        +boolean savePerson(PersonModel)
        +PersonModel updatePerson(PersonModel)
    }
    class PersonController {
        +void show()
    }
    class PersonApplication {
        +void main(String[])
    }
    class PersonModel {
        +Person currentPerson
        +List<Person> persons
    }
    class Person {
        +String toString()
        +String address
        +String city
        +String firstName
        +Long id
        +String lastName
        +String tel
        +Long zip
    }
    class Dao {
        +boolean create(T)
        +boolean delete(long)
        +List<T> readAll()
        +T readForIdentity(Long)
        +boolean update(T)
    }
    class AbstractDao {
        +void closeConnection()
        +Connection createConnection()
    }
    class PersonDao {
    }
    class PersonDaoJdbc {
        +boolean create(Person)
        +boolean delete(long)
        +List<Person> readAll()
        +Person readForIdentity(Long)
        +boolean update(Person)
    }
    class DaoFactory {
        +PersonDao createPersonDao()
    }

    BindableView ..> PersonOverViewFrame
    PersonOverViewFrame ..> PersonTableModel
    PersonLogic ..> PersonLogicImpl
    PersonController ..> PersonApplication
    PersonModel ..> Person
    Dao ..> AbstractDao
    AbstractDao <|-- PersonDao
    AbstractDao <|-- PersonDaoJdbc
    DaoFactory ..> PersonDao
    PersonDaoJdbc ..> PersonDao
    
```

The diagram illustrates the architecture of a person management application. It features several classes and their interactions:

- BindableView**: A base view class with methods `bindModel(PersonModel)` and `fillModel(PersonModel)`.
- PersonOverViewFrame**: A concrete view class that inherits from **BindableView**. It contains methods for binding, clearing fields, closing dialogs, filling the model, and showing various dialogs. It also has listeners for person addition, removal, and updates.
- PersonTableModel**: A data model class that provides methods for getting column names, values at specific indices, and column/row counts.
- PersonLogic**: An interface for business logic with methods `deletePerson`, `readAllPersons`, `savePerson`, and `updatePerson`.
- PersonLogicImpl**: A concrete implementation of **PersonLogic**.
- PersonController**: A controller class with a `show()` method.
- PersonApplication**: The main application class with a `main` method.
- PersonModel**: A model class representing a person, with attributes `currentPerson` and `persons` (a list of **Person** objects).
- Person**: A domain object representing a person with attributes `address`, `city`, `firstName`, `id`, `lastName`, `tel`, and `zip`.
- Dao**: A Data Access Object interface with methods `create`, `delete`, `readAll`, `readForIdentity`, and `update`.
- AbstractDao**: An abstract implementation of **Dao** with methods `closeConnection` and `createConnection`.
- PersonDao**: A concrete implementation of **Dao** for person data.
- PersonDaoJdbc**: A concrete implementation of **PersonDao** using JDBC.
- DaoFactory**: A factory class with a `createPersonDao` method.

Relationships are shown with solid lines (inheritance or association) and dashed lines (generalization or dependency). The **PersonDaoJdbc** class is a concrete implementation of the **PersonDao** interface, which in turn implements the **Dao** interface.

## Source-Code

Da die JDBC Schnittstelle im letzten Beispiel bereits behandelt wurde und nicht verändert wurde, wird dieser Teil hier ausgelassen und auf die neuen Klassen fokussiert.

### PersonApplication.java

```
package swp4.ue05.part1.ui;

import swp4.ue05.part1.ui.controller.PersonController;

import javax.swing.*;

public class PersonApplication {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                PersonController controller = new PersonController();
                controller.show();
            }
        });
    }
}
```

### PersonController.java

```
package swp4.ue05.part1.ui.controller;

import swp4.ue05.part1.logic.PersonLogic;
import swp4.ue05.part1.logic.impl.PersonLogicImpl;
import swp4.ue05.part1.ui.model.PersonModel;
import swp4.ue05.part1.ui.view.impl.PersonOverViewFrame;

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

public class PersonController {
    private PersonOverViewFrame view;
    private PersonModel model;
    private PersonLogic personLogic = new PersonLogicImpl();

    public PersonController() {
        this.view = new PersonOverViewFrame();
        this.model = personLogic.readAllPersons(new PersonModel());
        // add all button action listeners
        this.view.setPersonAddListener(new PersonAddListener());
        this.view.setPersonUpdateListener(new PersonUpdateListener());
        this.view.setPersonRemoveListener(new PersonRemoveListener());
        this.view.bindModel(this.model);
    }

    public void show() {
        view.setVisible(true);
    }

    public class PersonAddListener implements ActionListener {
```

```

@Override
public void actionPerformed(ActionEvent e) {
    // get infos from view
    model = view.fillModel(model);
    new Thread(this::addPerson).start();
    view.closeAddPersonDialog();
}

private void addPerson() {
    // save the person in the db, if successful read all persons and
return them
    if(personLogic.savePerson(model)) {
        model = personLogic.readAllPersons(model);
        SwingUtilities.invokeLater(() -> view.bindModel(model));
    } else {
        SwingUtilities.invokeLater(() -> view.showError("Could not
save person."));
    }
}

public class PersonUpdateListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        model = view.fillModel(model);
        if(model.getCurrentPerson().getId() >= 0) {
            new Thread(this::updatePerson).start();
            view.closeUpdatePersonDialog();
        } else {
            SwingUtilities.invokeLater(() -> view.showError("Can't update
person. No item selected."));
        }
    }

    private void updatePerson() {
        // if updating the person was successful, update the view
        if(personLogic.updatePerson(model) != null) {
            model = personLogic.readAllPersons(model);
            SwingUtilities.invokeLater(() -> view.bindModel(model));
        } else {
            SwingUtilities.invokeLater(() -> view.showError("Could not
update person."));
        }
    }
}

public class PersonRemoveListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        model = view.fillModel(model);
        if(model.getCurrentPerson().getId() >= 0) {
            new Thread(this::removePerson).start();
            view.closeRemovePersonDialog();
        } else {
            SwingUtilities.invokeLater(() -> view.showError("Can't update
person. No item selected."));
        }
    }
}

```

```

    }
}

private void removePerson() {
    // if removing the person was successful, update the view
    if(personLogic.deletePerson(model)) {
        model = personLogic.readAllPersons(model);
        SwingUtilities.invokeLater(() -> view.bindModel(model));
    } else {
        SwingUtilities.invokeLater(() -> view.showError("Could not
remove person."));
    }
}
}
}

```

### PersonModel.java

```

package swp4.ue05.part1.ui.model;

import swp4.ue05.part1.jdbc.domain.Person;

import java.util.ArrayList;
import java.util.List;

public class PersonModel {
    private Person currentPerson;
    private List<Person> persons;

    public PersonModel(Person currentPerson, List<Person> persons) {
        this.currentPerson = currentPerson;
        this.persons = persons;
    }

    public PersonModel() {
        currentPerson = null;
        persons = new ArrayList<>();
    }

    public Person getCurrentPerson() {
        return currentPerson;
    }

    public void setCurrentPerson(Person currentPerson) {
        this.currentPerson = currentPerson;
    }

    public List<Person> getPersons() {
        return persons;
    }

    public void setPersons(List<Person> persons) {
        this.persons = persons;
    }
}

```

## PersonLogicImpl.java

```
package swp4.ue05.part1.logic;

import swp4.ue05.part1.ui.model.PersonModel;

public interface PersonLogic {
    boolean savePerson(PersonModel model);
    PersonModel readAllPersons(PersonModel model);
    boolean deletePerson(PersonModel model);
    PersonModel updatePerson(PersonModel model);
}
```

## PersonLogic.java

```
package swp4.ue05.part1.logic.impl;

import swp4.ue05.part1.jdbc.dao.DaoFactory;
import swp4.ue05.part1.jdbc.dao.PersonDao;
import swp4.ue05.part1.logic.PersonLogic;
import swp4.ue05.part1.ui.model.PersonModel;

public class PersonLogicImpl implements PersonLogic {

    private PersonDao personDao = DaoFactory.createPersonDao();

    @Override
    public boolean savePerson(PersonModel model) {
        // insert only if model and currentPerson aren't NULL
        return model != null && model.getCurrentPerson() != null &&
personDao.create(model.getCurrentPerson());
    }

    @Override
    public PersonModel readAllPersons(PersonModel model) {
        return new PersonModel(model.getCurrentPerson(), personDao.readAll());
    }

    @Override
    public boolean deletePerson(PersonModel model) {
        return model != null && model.getCurrentPerson() != null &&
personDao.delete(model.getCurrentPerson().getId());
    }

    @Override
    public PersonModel updatePerson(PersonModel model) {
        if(personDao.update(model.getCurrentPerson())) {
            return new
PersonModel(personDao.readForIdentity(model.getCurrentPerson().getId()),
model.getPersons());
        }
        return null;
    }
}
```



## PersonTableModel.java

```
package swp4.ue05.part1.ui.view.impl;

import swp4.ue05.part1.jdbc.domain.Person;
import javax.swing.table.AbstractTableModel;
import java.util.List;

public class PersonTableModel extends AbstractTableModel {

    private List<Person> persons;
    // define table headers
    private static final String[] COLUMN_NAMES = {"#", "First Name", "Last Name", "City", "ZIP", "Address", "Telephone"};

    public PersonTableModel(List<Person> persons) {
        this.persons = persons;
    }

    @Override
    public String getColumnName(int column) {
        return COLUMN_NAMES[column];
    }

    @Override
    public int getRowCount() {
        return persons != null ? persons.size() : 0;
    }

    @Override
    public int getColumnCount() {
        return COLUMN_NAMES.length;
    }

    // make sure the tableModel returns the right values
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if (persons != null) {
            Person person = persons.get(rowIndex);
            if (person != null) {
                switch (columnIndex) {
                    case 0:
                        return person.getId();
                    case 1:
                        return person.getFirstName();
                    case 2:
                        return person.getLastName();
                    case 3:
                        return person.getCity();
                    case 4:
                        return person.getZip();
                    case 5:
                        return person.getAddress();
                    case 6:
                        return person.getTel();
                    default:
                        return null;
                }
            }
        }
    }
}
```

```

    }
    return null;
}
}

```

### BindableView.java

```

package swp4.ue05.part1.ui.view;

import swp4.ue05.part1.ui.model.PersonModel;

public interface BindableView {

    void bindModel(PersonModel model);

    PersonModel fillModel(PersonModel model);
}

```

### PersonOverview.java

```

package swp4.ue05.part1.ui.view.impl;

import swp4.ue05.part1.jdbc.domain.Person;
import swp4.ue05.part1.ui.controller.PersonController;
import swp4.ue05.part1.ui.model.PersonModel;
import swp4.ue05.part1.ui.view.BindableView;

import javax.swing.*;
import javax.swing.border.TitledBorder;
import java.awt.*;

public class PersonOverviewFrame extends JFrame implements BindableView {

    private JButton btnAddPerson;
    private JButton btnUpdatePerson;
    private JButton btnRemovePerson;
    private JTable tblPersons;

    private JDialog dlgAddPerson;
    private JDialog dlgUpdatePerson;
    private JDialog dlgRemovePerson;

    private JTextField txtFirstName;
    private JTextField txtLastName;
    private JTextField txtCity;
    private JTextField txtZip;
    private JTextField txtAddress;
    private JTextField txtTel;

    private PersonController.PersonAddListener personAddListener;
    private PersonController.PersonUpdateListener personUpdateListener;
    private PersonController.PersonRemoveListener personRemoveListener;

    private boolean removing;

    public PersonOverviewFrame() {initLayout();}

    public void setPersonAddListener(PersonController.PersonAddListener

```

```
personAddListener) {
    this.personAddListener = personAddListener;
}

    public void setPersonUpdateListener(PersonController.PersonUpdateListener
personUpdateListener) {
        this.personUpdateListener = personUpdateListener;
    }

    public void setPersonRemoveListener(PersonController.PersonRemoveListener
personRemoveListener) {
        this.personRemoveListener = personRemoveListener;
    }

    private void initLayout() {
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.setSize(600,200);
        this.setTitle("Person Application");

        this.getContentPane().setLayout(new BorderLayout(6,6));

        //region Header
        JPanel pnlHeader = new JPanel();
        pnlHeader.setLayout(new FlowLayout(FlowLayout.LEFT));
        JLabel lblTitle = new JLabel("Person Overview");
        pnlHeader.add(lblTitle);

        this.getContentPane().add(pnlHeader, BorderLayout.NORTH);
        //endregion

        //region Center - Table
        JPanel pnlCenter = new JPanel();
        TitledBorder tb = new TitledBorder("Available Persons");
        pnlCenter.setBorder(tb);

        pnlCenter.setLayout(new BorderLayout(6,6));
        tblPersons = new JTable();
        JScrollPane scrollPane = new JScrollPane(tblPersons);
        pnlCenter.add(scrollPane, BorderLayout.CENTER);

        this.getContentPane().add(pnlCenter, BorderLayout.CENTER);
        //endregion

        //region Footer
        JPanel pnlFooter = new JPanel();
        pnlFooter.setLayout(new FlowLayout(FlowLayout.RIGHT));
        btnAddPerson = new JButton("Add Person");
        btnAddPerson.addActionListener(e -> showAddPersonDialog());
        pnlFooter.add(btnAddPerson);

        btnUpdatePerson = new JButton("Update Person");
        btnUpdatePerson.addActionListener(e -> showUpdatePersonDialog());
        pnlFooter.add(btnUpdatePerson);

        btnRemovePerson = new JButton("Remove Person");
        btnRemovePerson.addActionListener(e -> showRemovePersonDialog());
        pnlFooter.add(btnRemovePerson);
    }
}
```

```
        this.getContentPane().add(pnlFooter, BorderLayout.SOUTH);
        //endregion
    }

    private void showAddPersonDialog() {
        removing = false;

        // Initialize Dialog
        dlgAddPerson = new JDialog(this, "Add Person", true);
        dlgAddPerson.setSize(new Dimension(400,150));
        dlgAddPerson.setLocationRelativeTo(btnAddPerson);
        dlgAddPerson.setResizable(false);

        JPanel pnlForm = new JPanel();
        TitledBorder tb = new TitledBorder("Add new person");
        pnlForm.setBorder(tb);
        pnlForm.setLayout(new GridLayout(3,2));

        // Initialize fields and add them to the panel
        txtFirstName = new JTextField();
        txtLastName = new JTextField();
        txtCity = new JTextField();
        txtZip = new JTextField();
        txtAddress = new JTextField();
        txtTel = new JTextField();

        pnlForm.add(new JLabel("First Name: "));
        pnlForm.add(txtFirstName);
        pnlForm.add(new JLabel("Last Name: "));
        pnlForm.add(txtLastName);
        pnlForm.add(new JLabel("City: "));
        pnlForm.add(txtCity);
        pnlForm.add(new JLabel("ZIP: "));
        pnlForm.add(txtZip);
        pnlForm.add(new JLabel("Address: "));
        pnlForm.add(txtAddress);
        pnlForm.add(new JLabel("Tel.: "));
        pnlForm.add(txtTel);

        // add the button to the bottom of the panel
        JPanel pnlFooter = new JPanel();
        pnlFooter.setLayout(new FlowLayout(FlowLayout.RIGHT));
        JButton btnSavePerson = new JButton("Save Person");
        btnSavePerson.addActionListener(personAddListener);
        pnlFooter.add(btnSavePerson);

        // finally, add the panel to the dialog
        dlgAddPerson.getContentPane().add(pnlForm, BorderLayout.CENTER);
        dlgAddPerson.getContentPane().add(pnlFooter, BorderLayout.SOUTH);

        dlgAddPerson.setVisible(true);
        dlgAddPerson.pack();
    }

    public void closeAddPersonDialog() {
        clearFields();
        dlgAddPerson.setVisible(false);
    }
}
```

```
private void showUpdatePersonDialog() {
    removing = false;
    int row = tblPersons.getSelectedRow();
    if(row < 0) {
        showError("Please select a person to update.");
        return;
    }

    // dialog initialization
    dlgUpdatePerson = new JDialog(this, "Update Person", true);
    dlgUpdatePerson.setSize(new Dimension(400,150));
    dlgUpdatePerson.setLocationRelativeTo(btnUpdatePerson);
    dlgUpdatePerson.setResizable(false);

    JPanel pnlForm = new JPanel();
    TitledBorder tb = new TitledBorder("Update person");
    pnlForm.setBorder(tb);
    pnlForm.setLayout(new GridLayout(3,2));

    // get data from table
    Long id = (Long) tblPersons.getValueAt(row, 0);
    txtFirstName = new JTextField(tblPersons.getValueAt(row,
1) .toString());
    txtLastName = new JTextField(tblPersons.getValueAt(row,
2) .toString());
    txtCity = new JTextField(tblPersons.getValueAt(row, 3).toString());
    txtZip = new JTextField(tblPersons.getValueAt(row, 4).toString());
    txtAddress = new JTextField(tblPersons.getValueAt(row, 5).toString());
    txtTel = new JTextField(tblPersons.getValueAt(row, 6).toString());

    // show Labels and prefilled textboxes
    pnlForm.add(new JLabel("First Name: "));
    pnlForm.add(txtFirstName);
    pnlForm.add(new JLabel("Last Name: "));
    pnlForm.add(txtLastName);
    pnlForm.add(new JLabel("City: "));
    pnlForm.add(txtCity);
    pnlForm.add(new JLabel("ZIP: "));
    pnlForm.add(txtZip);
    pnlForm.add(new JLabel("Address: "));
    pnlForm.add(txtAddress);
    pnlForm.add(new JLabel("Tel.: "));
    pnlForm.add(txtTel);

    // add footer with button
    JPanel pnlFooter = new JPanel();
    pnlFooter.setLayout(new FlowLayout(FlowLayout.RIGHT));
    JButton btnSavePerson = new JButton("Save Person");
    btnSavePerson.addActionListener(personUpdateListener);
    pnlFooter.add(btnSavePerson);

    dlgUpdatePerson.getContentPane().add(pnlForm, BorderLayout.CENTER);
    dlgUpdatePerson.getContentPane().add(pnlFooter, BorderLayout.SOUTH);

    dlgUpdatePerson.setVisible(true);
    dlgUpdatePerson.pack();
}
```

```
public void closeUpdatePersonDialog() {
    clearFields();
    dlgUpdatePerson.setVisible(false);
}

private void showRemovePersonDialog() {
    int row = tblPersons.getSelectedRow();
    if(row < 0) {
        showError("Please select a person to remove.");
        return;
    }
    removing = true;
    // init dialog
    dlgRemovePerson = new JDialog(this, "Remove Person", true);
    dlgRemovePerson.setSize(new Dimension(400,150));
    dlgRemovePerson.setLocationRelativeTo(btnRemovePerson);
    dlgRemovePerson.setResizable(false);

    JPanel pnlForm = new JPanel();
    TitledBorder tb = new TitledBorder("Remove person");
    pnlForm.setBorder(tb);
    pnlForm.setLayout(new GridLayout(3,2));

    // add removal info and button
    JLabel lblRemovePerson = new JLabel("Do you really want to remove the
selected person?");
    pnlForm.add(lblRemovePerson);

    JPanel pnlFooter = new JPanel();
    pnlFooter.setLayout(new FlowLayout(FlowLayout.RIGHT));
    JButton btnRemovePerson = new JButton("Remove Person");
    btnRemovePerson.addActionListener(personRemoveListener);
    pnlFooter.add(btnRemovePerson);

    dlgRemovePerson.getContentPane().add(pnlForm, BorderLayout.CENTER);
    dlgRemovePerson.getContentPane().add(pnlFooter, BorderLayout.SOUTH);

    dlgRemovePerson.setVisible(true);
    dlgRemovePerson.pack();
}

public void closeRemovePersonDialog() {
    dlgRemovePerson.setVisible(false);
}

private void clearFields() {
    txtFirstName.setText("");
    txtLastName.setText("");
    txtCity.setText("");
    txtZip.setText("");
    txtAddress.setText("");
    txtTel.setText("");
}

public void showError(String message) {
    // show a delegated error
    JOptionPane.showMessageDialog(this, message, "Error",
JOptionPane.ERROR_MESSAGE);
}
```

```

@Override
public void bindModel(PersonModel model) {
    // set new model as data and display it
    tblPersons.setModel(new PersonTableModel(model.getPersons()));
}

@Override
public PersonModel fillModel(PersonModel model) {

    Long id = null;
    String firstName = "";
    String lastName = "";
    String city = "";
    Long zip = null;
    String address = "";
    String tel = "";

    // if a row is selected in the table, get the persons index for
removal
    int rowIndex = tblPersons.getSelectedRow();
    if(rowIndex >= 0) {
        id = (Long)tblPersons.getValueAt(rowIndex,0);
    }

    // fill model from dialog fields if mode is not "removing"
    if(!removing) {
        firstName = txtFirstName.getText();
        lastName = txtLastName.getText();
        city = txtCity.getText();
        try {
            zip = Long.parseLong(txtZip.getText());
        } catch(NumberFormatException e) {
            zip = null;
        }
        address = txtAddress.getText();
        tel = txtTel.getText();
    }

    // check for false inputs and show errors
    if(removing && id == null) {
        showError("Error removing person. Please select an entry.");
    } else if (!removing && firstName.isEmpty()) {
        showError("Invalid firstname.");
    } else if (!removing && lastName.isEmpty()) {
        showError("Invalid lastname.");
    } else if (!removing && city.isEmpty()) {
        showError("Invalid city.");
    } else if (!removing && zip == null) {
        showError("Invalid zip code.");
    } else if (!removing && address.isEmpty()) {
        showError("Invalid address.");
    } else if (!removing && tel.isEmpty()) {
        showError("Invalid telephone number.");
    } else {
        // if no errors are present, create a new person using the read
data
        Person newPerson = new

```

```
Person(firstName,lastName,city,zip,address,tel);
    newPerson.setId(id);

    // ... and set the currentPerson and return the model
    model.setCurrentPerson(newPerson);
}

return model;
}
```



## Testfälle

Zusätzlich zu den Unit-Tests aus Übung 4, folgen noch weitere Testfälle:

- Nichtstarten der Datenbank
- Leere Input-Felder
- Falsches Input-Format (String statt Zahl für PLZ)
- Update ohne Auswahl eines Eintrags
- Löschen ohne Auswahl eines Eintrags

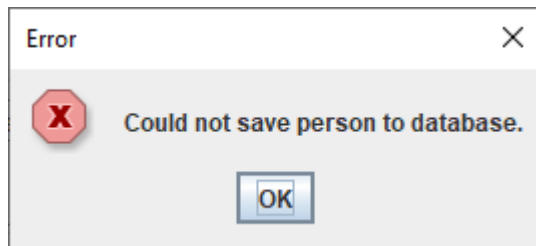
### Nichtstarten der Datenbank

Das Programm startet zwar, aber ist logischerweise nicht benutzbar.

Output:

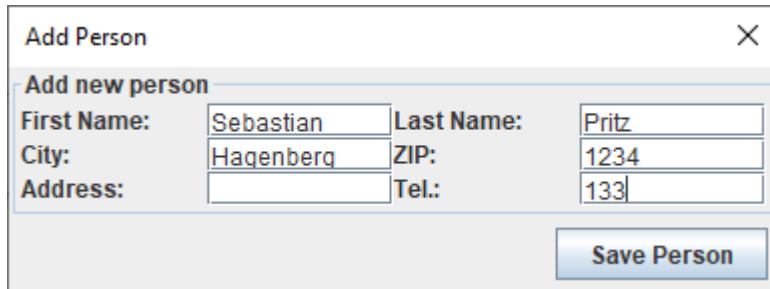
```
Failed to fetch all persons.  
java.sql.SQLNonTransientConnectionException: java.net.ConnectException: Fehler beim  
Herstellen der Verbindung zu Server localhost auf Port 1.527 mit Meldung Connection  
refused: connect.
```

Wird dennoch versucht die Funktionalitäten zu verwenden, so kommt nach kurzer Zeit ein Error:

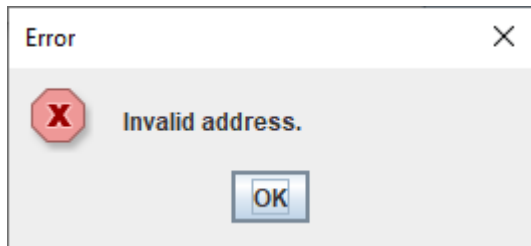


### Leere Input-Felder

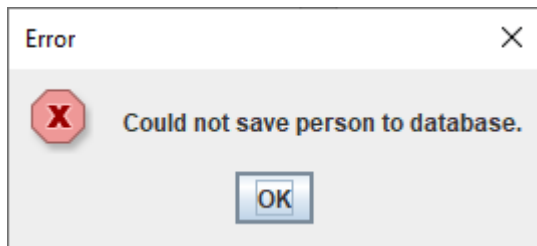
Wird der „Save Person“ Knopf gedrückt, bevor ALLE Felder ausgefüllt wurden, so kommt es zu einem Fehler:



A dialog box titled "Add Person" with a close button (X) in the top right corner. It contains a section titled "Add new person" with four input fields: "First Name:" (Sebastian), "Last Name:" (Pritz), "City:" (Hagenberg), and "ZIP:" (1234). Below these are two more fields: "Address:" (empty) and "Tel.:" (133). A "Save Person" button is at the bottom right.



An error dialog box titled "Error" with a close button (X) in the top right corner. It features a red octagonal icon with a white 'X' and the text "Invalid address." Below the message is an "OK" button.

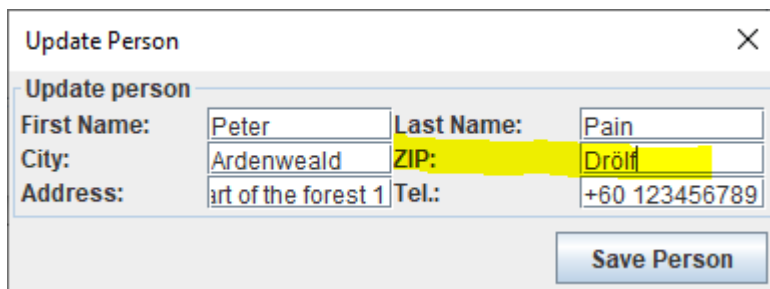


An error dialog box titled "Error" with a close button (X) in the top right corner. It features a red octagonal icon with a white 'X' and the text "Could not save person to database." Below the message is an "OK" button.

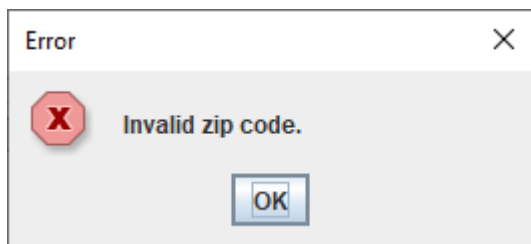
Das lässt sich natürlich für alle anderen Felder wiederholen und verhält sich ähnlich.

### Falsches Input-Format

Bei Eingabe der Postleitzahl kann es zu Problemen kommen, da diese als Zahl (Long) gespeichert wird. Dieses Problem wird abgefangen und als Error kommuniziert:



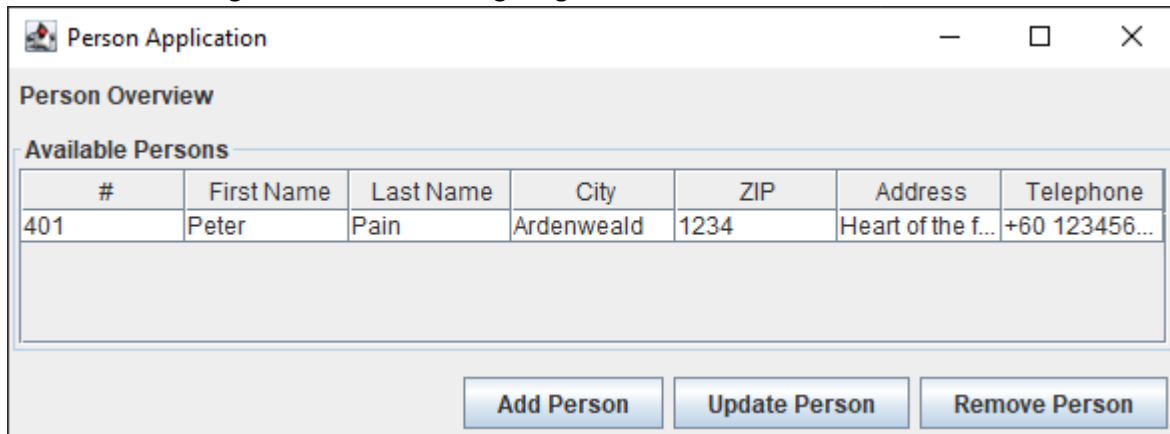
A dialog box titled "Update Person" with a close button (X) in the top right corner. It contains a section titled "Update person" with four input fields: "First Name:" (Peter), "Last Name:" (Pain), "City:" (Ardenweald), and "ZIP:" (Dröfl). Below these are two more fields: "Address:" (art of the forest 1) and "Tel.:" (+60 123456789). A "Save Person" button is at the bottom right.



An error dialog box titled "Error" with a close button (X) in the top right corner. It features a red octagonal icon with a white 'X' and the text "Invalid zip code." Below the message is an "OK" button.

## Update ohne Auswahl eines Eintrags

Beim Start des Programms ist kein Eintrag ausgewählt.

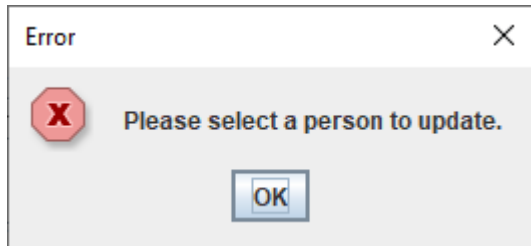


The screenshot shows a window titled "Person Application" with a "Person Overview" section. Below this is a table titled "Available Persons" with the following data:

#	First Name	Last Name	City	ZIP	Address	Telephone
401	Peter	Pain	Ardenweald	1234	Heart of the f...	+60 123456...

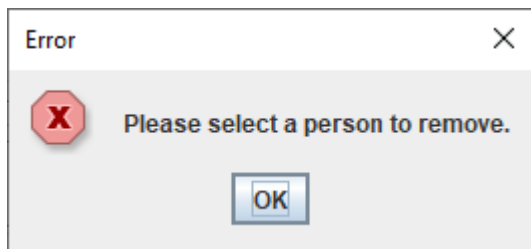
Below the table are three buttons: "Add Person", "Update Person", and "Remove Person".

Wird direkt „Update“ ausgewählt, so ist kein Eintrag (und somit keine ID!) vorhanden und der Prozess wird abgebrochen.



## Löschen ohne Auswahl eines Eintrags

Ähnlich verhält es sich mit dem „Remove“-Button.



## Teil 2 – Grafische Benutzerschnittstelle für Alignments

Im zweiten Teil dieses Übungszettels geht es um die Implementierung eines GUIs für Alignments. Sequenzen, Scoring Metrics, sowie der Alignment-Algorithmus sollen ausgewählt werden können und daraus Ergebnisse berechnet werden und angezeigt werden.

### Lösungsidee

Ich würde mich nach wie vor wieder am MVC-Design orientieren

- Der View stellt alle Menüs, Anzeigen und Buttons zur Verfügung
- Der Controller verarbeitet die datenbezogenen Befehle über ActionListener von den Buttons und durch Abfrage der Textfelder im UI
- Das Model beinhaltet mehrere Alignment-Einstellungen, die sich aus den Sequenzen, den Scoring-Metriken und dem Algorithmus zusammensetzen. Zusätzlich würde ich noch eine eindeutige ID vergeben.
- Ich bin mir nicht sicher, ob die „Logik“-Klasse wie im vorherigen Beispiel benötigt wird, oder ob diese nur bei Datenbankanwendungen zur Anwendung kommt. Grundsätzlich würde ich diese aber verwenden, um den Zugriff auf das Model zu streamlinen, sodass alle datenbezogenen Nullpointer-Vergleiche etc. wie gehabt hier abgefangen werden und die Elemente ins Model eingefügt werden.

Beim UI würde ich grundsätzlich so vorgehen, dass ich das About-Menü in der Menüleiste als Menüpunkt implementiere. Ich schätze, dass sich diese ähnlich zu Buttons verhalten und über einen ActionListener ein neuer Dialog geöffnet wird, der etwas anzeigt.

Das restliche UI bietet sich gut an, um es in 3 Teile zu Teilen und ein BorderLayout zu verwenden:

- Links oben: Panel für gespeicherte Alignments und die zugehörigen Buttons
- Rechts oben (CENTER, sodass es mitskaliert): Input-Felder für Sequenzen, Scoring-Metriken etc. Hierfür eignet sich ein GridLayout innerhalb des Panels, damit die Spalten gleich groß bleiben bzw. die Anordnung der Elemente besser passt.
  - o Nachtrag: es wurde schlussendlich ein GridBagLayout verwendet, da dieses wesentlich flexibler ist, was spaltenübergreifende Elemente und Skalierung angeht
- Unten (FOOTER): Panel mit Textarea für Ergebnis

Für die Speicherung der Daten würde sich nun eine Liste besser eignen, da nur noch eine Spalte benötigt wird.

Im rechten Teil werden wie gesagt die Daten eingegeben und beim Druck auf „New“ als Alignment-Einstellung in der Liste gespeichert. Wählt man den Eintrag in der Liste aus, so sollen die Felder wieder entsprechend befüllt werden.

Mir war die Funktion des Compute-Button nicht ganz klar, da das Ergebnis ja direkt beim Auswählen einer Einstellung aktualisiert werden soll. Der Button-Compute erlaubt in meiner Implementierung somit das einmalige bzw. erstmalige Ausführen eines Alignments ohne dieses zu Speichern.

Der Button „Remove“ entfernt einfach den ausgewählten Eintrag aus der Liste und „Save“ speichert diesen als File.

\*Ich war übrigens so frei und habe ein MenuItem zu einem User-Guide implementiert.



## Source-Code

## AlignmentApplication.java

```
package swp4.ue05.part2;

import swp4.ue05.part2.ui.controller.AlignmentController;

import javax.swing.*.*;

public class AlignmentApplication {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                AlignmentController controller = new AlignmentController();
                controller.show();
            }
        });
    }
}
```

## AlignmentItem.java

```
package swp4.ue05.part2.domain;

import java.util.Objects;

public class AlignmentItem {

    // an alignmentItem contains all inserted values and a unique id
    private long id;
    private String sequence1;
    private String sequence2;
    private int match;
    private int mismatch;
    private int gap;
    private int algorithm;
    private static long currentid = 0;

    public AlignmentItem(String sequence1, String sequence2, int match, int mismatch, int gap, int algorithm) {
        id = ++currentid;
        this.sequence1 = sequence1;
        this.sequence2 = sequence2;
        this.match = match;
        this.mismatch = mismatch;
        this.gap = gap;
        this.algorithm = algorithm;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }
}
```

```
public String getSequence1() {
    return sequence1;
}

public void setSequence1(String sequence1) {
    this.sequence1 = sequence1;
}

public String getSequence2() {
    return sequence2;
}

public void setSequence2(String sequence2) {
    this.sequence2 = sequence2;
}

public int getMatch() {
    return match;
}

public void setMatch(int match) {
    this.match = match;
}

public int getMismatch() {
    return mismatch;
}

public void setMismatch(int mismatch) {
    this.mismatch = mismatch;
}

public int getGap() {
    return gap;
}

public void setGap(int gap) {
    this.gap = gap;
}

public int getAlgorithm() {
    return algorithm;
}

public void setAlgorithm(int algorithm) {
    this.algorithm = algorithm;
}

@Override
public String toString() {
    // show the alignments id, the first 10 characters of each sequence as
    // well as the scoring
    return "Alignment "+id+": "+(sequence1.length() <= 10 ? sequence1 :
sequence1.substring(0,10)) + " :: " + (sequence2.length() <= 10 ? sequence2 :
sequence2.substring(0,10)) + " " + match + "/" + mismatch + "/" + gap;
}
```

```

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    AlignmentItem that = (AlignmentItem) o;
    return id == that.id;
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
}

```

### BindableView.java

```

package swp4.ue05.part2.ui.view;

import swp4.ue05.part2.ui.model.AlignmentModel;

public interface BindableView {
    void bindModel(AlignmentModel model);

    AlignmentModel fillModel(AlignmentModel model);
}

```

### AlignmentOverViewFrame.java

```

package swp4.ue05.part2.ui.view.impl;

import chris.dj.nativeswing.swtimpl.NativeInterface;
import chris.dj.nativeswing.swtimpl.components.JWebBrowser;
import neobio.alignment.*;
import swp4.ue05.part2.domain.AlignmentItem;
import swp4.ue05.part2.ui.controller.AlignmentController;
import swp4.ue05.part2.ui.view.BindableView;
import swp4.ue05.part2.ui.model.AlignmentModel;

import javax.swing.*;
import javax.swing.border.TitledBorder;
import javax.swing.event.ListSelectionListener;
import java.awt.*;
import java.io.*;
import java.text.ParseException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class AlignmentOverViewFrame extends JFrame implements BindableView {

    //region UI-Elements
    JDialog dlgUserguide;

    JMenuBar menuBar;
    JMenu menuHelp;
    JMenuItem menuHelpAbout;
    JMenuItem menuHelpUserguide;
}

```



```
JList<AlignmentItem> listAlignments;
JTextArea txtResult;
JTextArea txtSeq1;
JTextArea txtSeq2;
JSpinner spnMatch;
JSpinner spnMismatch;
JSpinner spnGap;
JComboBox<String> cobAlgorithm;

JButton btnNew;
JButton btnRemove;
JButton btnCompute;
JButton btnSave;
JButton btnLoad1;
JButton btnLoad2;
//endregion

NeedlemanWunsch needlemanWunsch = new NeedlemanWunsch();
SmithWaterman smithWaterman = new SmithWaterman();

public AlignmentOverviewFrame() {
    initLayout();
}

public void
setAlignmentAddListener(AlignmentController.AlignmentAddListener
alignmentAddListener) {
    this.btnNew.addActionListener(alignmentAddListener);
}

public void
setAlignmentRemoveListener(AlignmentController.AlignmentRemoveListener
alignmentRemoveListener) {
    this.btnRemove.addActionListener(alignmentRemoveListener);
}

public void setListSelectionListener(ListSelectionListener
listSelectionListener) {
    this.listAlignments.addListSelectionListener(listSelectionListener);
}

private void initLayout() {
    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    this.setSize(1280, 720);
    this.setMinimumSize(new Dimension(800, 600));
    this.setTitle("Alignments");
    this.getContentPane().setLayout(new BorderLayout(6, 6));

    //region Menubar
    menuBar = new JMenuBar();
    menuHelp = new JMenu("Help");
    menuBar.add(menuHelp);
    menuHelpAbout = new JMenuItem("About");
    menuHelpAbout.addActionListener(e -> showAbout());
    menuHelp.add(menuHelpAbout);
    menuHelpUserguide = new JMenuItem("User-Guide");
    menuHelpUserguide.addActionListener(e -> showUserGuide());
```

```
menuHelp.add(menuHelpUserguide);
this.setJMenuBar(menuBar);
//endregion

//region TablePanel
JPanel pnlTable = new JPanel();
pnlTable.setLayout(new BorderLayout(6,6));
TitledBorder tbTable = new TitledBorder("Alignments");
pnlTable.setBorder(tbTable);

listAlignments = new JList();
JScrollPane scrollPane = new JScrollPane(listAlignments);
pnlTable.add(scrollPane, BorderLayout.CENTER);

// Buttons beneath table
JPanel pnlTableButtons = new JPanel();
pnlTableButtons.setLayout(new FlowLayout(FlowLayout.RIGHT));

btnNew = new JButton("New");
pnlTableButtons.add(btnNew);

btnRemove = new JButton("Remove");
pnlTableButtons.add(btnRemove);

btnCompute = new JButton("Compute");
btnCompute.addActionListener(e -> computeResult());
pnlTableButtons.add(btnCompute);

btnSave = new JButton("Save result...");
btnSave.addActionListener(e -> showSaveDialog());
pnlTableButtons.add(btnSave);

pnlTable.add(pnlTableButtons, BorderLayout.SOUTH);
pnlTable.setPreferredSize(new Dimension(400,100));

this.getContentPane().add(pnlTable, BorderLayout.LINE_START);
//endregion

//region InputPanel
JPanel pnlInput = new JPanel();
TitledBorder tbInput = new TitledBorder("Alignment Command");
pnlInput.setBorder(tbInput);
GridBagLayout gbl = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
pnlInput.setLayout(gbl);

//region Line 1 - Sequence 1
gbc.insets = new Insets(3,3,3,3);
gbc.fill = GridBagConstraints.BOTH;
gbc.anchor = GridBagConstraints.WEST;

// used GridBagLayout --> specify grid coordinates, weight (for
scaling) and other settings for each element
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weightx = 0;
gbc.weighty = 0.5;
JLabel lblSeq1 = new JLabel("Sequence 1");
```

```
pnlInput.add(lblSeq1, gbc);

gbc.gridx = 1;
gbc.weightx = 1;

txaSeq1 = new JTextArea("");
txaSeq1.setLineWrap(true);
JScrollPane scrollPaneSeq1 = new JScrollPane(txaSeq1);
pnlInput.add(scrollPaneSeq1, gbc);

gbc.gridx = 2;
gbc.weightx = 0;
btnLoad1 = new JButton("Load...");
btnLoad1.setPreferredSize(new Dimension(100,100));
btnLoad1.addActionListener(e -> showFileDialog(1));
pnlInput.add(btnLoad1, gbc);
//endregion
//region Line 2 - Sequence 2
gbc.gridx = 0;
gbc.gridy = 1;
gbc.weightx = 0;
JLabel lblSeq2 = new JLabel("Sequence 2");
pnlInput.add(lblSeq2, gbc);

gbc.gridx = 1;
gbc.weightx = 1;
txaSeq2 = new JTextArea("");
txaSeq2.setLineWrap(true);
JScrollPane scrollPaneSeq2 = new JScrollPane(txaSeq2);
pnlInput.add(scrollPaneSeq2, gbc);

gbc.gridx = 2;
gbc.weightx = 0;
btnLoad2 = new JButton("Load...");
btnLoad2.setPreferredSize(new Dimension(100,100));
btnLoad2.addActionListener(e -> showFileDialog(2));
pnlInput.add(btnLoad2, gbc);
gbc.weighty = 0;
//endregion
//region Line 3 - Match
gbc.gridx = 0;
gbc.gridy = 2;
JLabel lblMatch = new JLabel("Match");
pnlInput.add(lblMatch, gbc);

gbc.gridx = 1;
gbc.weightx = 1;
gbc.gridwidth = 2;
spnMatch = new JSpinner(new SpinnerNumberModel(1,0,100,1));
pnlInput.add(spnMatch, gbc);
//endregion
//region Line 4 - Mismatch
gbc.gridx = 0;
gbc.gridy = 3;
gbc.weightx = 0;
JLabel lblMismatch = new JLabel("Mismatch");
pnlInput.add(lblMismatch, gbc);
```

```

        gbc.gridx = 1;
        gbc.weightx = 1;
        gbc.gridwidth = 2;
        spnMismatch = new JSpinner(new SpinnerNumberModel(-1,-100,0,1));
        pnlInput.add(spnMismatch,gbc);
        //endregion
        //region Line 5 - Gap
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.weightx = 0;
        JLabel lblGap = new JLabel("Gap");
        pnlInput.add(lblGap, gbc);

        gbc.gridx = 1;
        gbc.weightx = 1;
        gbc.gridwidth = 2;
        spnGap = new JSpinner(new SpinnerNumberModel(-1,-100,0,1));
        pnlInput.add(spnGap,gbc);
        //endregion
        //region Line 6 - Mismatch
        gbc.gridx = 0;
        gbc.gridy = 5;
        gbc.weightx = 0;
        JLabel lblAlgorithm = new JLabel("Algorithm");
        pnlInput.add(lblAlgorithm, gbc);

        gbc.gridx = 1;
        gbc.weightx = 1;
        gbc.gridwidth = 2;
        cobAlgorithm = new JComboBox<>();
        cobAlgorithm.addItem("NeedlemanWunsch");
        cobAlgorithm.addItem("SmithWaterman");
        pnlInput.add(cobAlgorithm,gbc);
        //endregion

        this.getContentPane().add(pnlInput, BorderLayout.CENTER);
        //endregion

        //region ResultPanel
        JPanel pnlResult = new JPanel();
        pnlResult.setLayout(new BorderLayout(6,6));
        TitledBorder tbResult = new TitledBorder("Result");
        pnlResult.setBorder(tbResult);

        txtaResult = new JTextArea();
        // set monospaced font to allow proper display for alignments
        txtaResult.setFont(new Font("monospaced", Font.PLAIN, 12));
        txtaResult.setEditable(false);
        txtaResult.setLineWrap(true);
        JScrollPane scrollPaneRes = new JScrollPane(txtaResult);
        pnlResult.add(scrollPaneRes, BorderLayout.CENTER);
        scrollPaneRes.setPreferredSize(new Dimension(200,200));
        this.getContentPane().add(pnlResult, BorderLayout.SOUTH);
        //endregion
    }

    public void changeSelection(AlignmentItem alignmentItem) {
        // when the selection changes, fill in all the fields to the items

```

```
settings
    if(alignmentItem != null) {
        txaseq1.setText(alignmentItem.getSequence1());
        txaseq2.setText(alignmentItem.getSequence2());
        spnMatch.setValue(alignmentItem.getMatch());
        spnMismatch.setValue(alignmentItem.getMismatch());
        spnGap.setValue(alignmentItem.getGap());
        cobAlgorithm.setSelectedIndex(alignmentItem.getAlgorithm());
    }
}

private void showAbout() {
    JOptionPane.showMessageDialog(this, "This program allows the
computation of alignments using a graphical interface.", "About",
JOptionPane.INFORMATION_MESSAGE);
}

private void showUserGuide() {
    //region userGuide
    NativeInterface.open();
    dlgUserguide = new JDialog(this, "User Guide", true);
    dlgUserguide.setSize(new Dimension(720,480));
    dlgUserguide.setLocationRelativeTo(menuHelpUserguide);
    dlgUserguide.setResizable(false);

    JPanel pnlGuide = new JPanel();
    TitledBorder tb = new TitledBorder("User guide");
    pnlGuide.setBorder(tb);
    pnlGuide.setLayout(new BorderLayout(6,6));

    JWebBrowser wb = new JWebBrowser();
    pnlGuide.add(wb, BorderLayout.CENTER);
    wb.setBarsVisible(false);
    wb.navigate("https://streamable.com/e/shil2");

    dlgUserguide.getContentPane().add(pnlGuide);
    dlgUserguide.setVisible(true);
    NativeInterface.close();
    //endregion
}

public void showError(String message) {
    JOptionPane.showMessageDialog(this, message, "Error",
JOptionPane.ERROR_MESSAGE);
}

private void showFileDialog(int fieldNum) {
    JFileChooser jfc = new JFileChooser();
    if(jfc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
        // if the filechoosers action was approved, get the filepath
        String filePath = jfc.getSelectedFile().toPath().toString();

        try(BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            String line;
            StringBuilder sb = new StringBuilder();
            // use the filepath and get file content
            while((line = reader.readLine()) != null) {
```

```
        sb.append(line);
    }

    // finally, set the file content as text of the sequences
    if(fieldNum == 1) {
        txaSeq1.setText(sb.toString());
    } else {
        txaSeq2.setText(sb.toString());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}

private void showSaveDialog() {
    JFileChooser jfc = new JFileChooser();
    if(jfc.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
        // if a location and name have been picked, get the path.
        String filePath = jfc.getSelectedFile().getPath().toString();

        // and write the result into the file
        try(FileWriter writer = new FileWriter(filePath)) {
            writer.write(txaResult.getText());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void computeResult() {
    // fetch all necessary values
    String seq1 = txaSeq1.getText();
    String seq2 = txaSeq2.getText();
    Pattern pattern = Pattern.compile("[^a-zA-Z]");
    Matcher matcherSeq1 = pattern.matcher(seq1);
    Matcher matcherSeq2 = pattern.matcher(seq2);

    // if sequences are malformed throw an error and stop
    if(matcherSeq1.find()) {
        showError("Sequence 1 contains invalid characters.");
    } else if (seq1.isEmpty()) {
        showError("Sequence 1 is empty.");
    } else if (matcherSeq2.find()) {
        showError("Sequence 2 contains invalid characters.");
    } else if (seq2.isEmpty()) {
        showError("Sequence 2 is empty.");
    } else {
        // if sequences are okay, get the scoring scheme
        try {
            spnMatch.commitEdit();
            spnMisMatch.commitEdit();
            spnGap.commitEdit();
        }
    }
}
```

```

        } catch (ParseException e) {
            e.printStackTrace();
        }
        BasicScoringScheme scoringScheme = new
BasicScoringScheme((int)spnMatch.getValue(), (int)spnMisMatch.getValue(),
(int)spnGap.getValue());
        PairwiseAlignment pairwiseAlignment = null;

        btnCompute.setEnabled(false);
        // finally, start the desired algorithm based on the combobox
selection
        try {
            switch (cobAlgorithm.getSelectedIndex()) {
                case 0:
                    needlemanWunsch.setScoringScheme(scoringScheme);
                    needlemanWunsch.loadSequences(new StringReader(seq1),
new StringReader(seq2));
                    pairwiseAlignment =
needlemanWunsch.getPairwiseAlignment();
                    break;
                case 1:
                    smithWaterman.setScoringScheme(scoringScheme);
                    smithWaterman.loadSequences(new
StringReader(txaSeq1.getText()), new StringReader(txaSeq2.getText()));
                    pairwiseAlignment =
needlemanWunsch.getPairwiseAlignment();
                    break;
            }

        } catch (IOException | InvalidSequenceException e) {
            e.printStackTrace();
        } catch (IncompatibleScoringSchemeException e) {
            e.printStackTrace();
        }

        // if the algorithm finished successfully, fill in the result
        if (pairwiseAlignment != null) {
            txaResult.setText(new StringBuilder()
                .append("#####\n")
                .append("Algorithm: " +
(cobAlgorithm.getSelectedIndex() == 0 ? "NeedlemanWunsch" : "SmithWaterman") +
"\n")
                .append("Score: " + pairwiseAlignment.getScore() +
"\n")
                .append("#####\n")
                .append("\n")
                .append(pairwiseAlignment)
                .toString()
            );
        } else {
            txaResult.setText("PairwiseAlignment couldn't be computed.");
        }
        btnCompute.setEnabled(true);
    }
}

public AlignmentModel fillModelFromSelection(AlignmentModel model) {
    // get the selected alignmentItem from the List

```

```
        model.setCurrentAlignment(listAlignments.getSelectedValue());
        return model;
    }

    @Override
    public void bindModel(AlignmentModel model) {
        listAlignments.setModel(new
AlignmentListModel(model.getAlignments()));
    }

    @Override
    public AlignmentModel fillModel(AlignmentModel model) {
        // fetch all values
        String seq1 = txaSeq1.getText();
        String seq2 = txaSeq2.getText();
        // commit changes for manually entered values
        try {
            spnMatch.commitEdit();
            spnMismatch.commitEdit();
            spnGap.commitEdit();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        int match = (int) spnMatch.getValue();
        int mismatch = (int) spnMismatch.getValue();
        int gap = (int) spnGap.getValue();
        int algorithm = cobAlgorithm.getSelectedIndex();

        Pattern pattern = Pattern.compile("[^a-zA-Z]");
        Matcher matcherSeq1 = pattern.matcher(seq1);
        Matcher matcherSeq2 = pattern.matcher(seq2);

        // check if sequences contain letters only and arent empty
        if (matcherSeq1.find()) {
            showError("Sequence 1 contains invalid characters.");
        } else if (seq1.isEmpty()) {
            showError("Sequence 1 is empty.");
        } else if (matcherSeq2.find()) {
            showError("Sequence 2 contains invalid characters.");
        } else if (seq2.isEmpty()) {
            showError("Sequence 2 is empty.");
        } else {
            // if requirements are met, set the data
            model.setCurrentAlignment(new AlignmentItem(seq1, seq2, match,
mismatch, gap, algorithm));
        }

        return model;
    }
}
```



## AlignmentListModel.java

```
package swp4.ue05.part2.ui.view.impl;

import swp4.ue05.part2.domain.AlignmentItem;

import javax.swing.*;
import java.util.List;

public class AlignmentListModel extends AbstractListModel<AlignmentItem> {

    private List<AlignmentItem> alignments;

    public AlignmentListModel(List<AlignmentItem> alignments) {
        this.alignments = alignments;
    }

    @Override
    public int getSize() {
        return alignments.size();
    }

    @Override
    public AlignmentItem getElementAt(int index) {
        if(index < getSize()) {
            return alignments.get(index);
        }
        return null;
    }
}
```

## AlignmentModel.java

```
package swp4.ue05.part2.ui.model;

import swp4.ue05.part2.domain.AlignmentItem;

import java.util.ArrayList;
import java.util.List;

public class AlignmentModel {

    private AlignmentItem currentAlignment;
    private List<AlignmentItem> alignments;

    public AlignmentModel(AlignmentItem currentAlignment, List<AlignmentItem>
alignments) {
        this.currentAlignment = currentAlignment;
        this.alignments = alignments;
    }

    public AlignmentModel() {
        this.currentAlignment = null;
        this.alignments = new ArrayList<>();
    }

    public AlignmentItem getCurrentAlignment() {
        return currentAlignment;
    }
}
```



```

        AlignmentItem selectedValue = (AlignmentItem)
list.getSelectedValue();
        view.changeSelection(selectedValue);
    }
}
});
this.view.bindModel(this.model);
}

public void show() {
    this.view.setVisible(true);
}

// add action listeners for remove and add process
public class AlignmentAddListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        model = view.fillModel(model);
        new Thread(this::addAlignment).start();
    }

    private void addAlignment() {
        if(alignmentLogic.addAlignment(model)) {
            model = alignmentLogic.getAlignments(model);
            SwingUtilities.invokeLater(() -> view.bindModel(model));
        } else {
            SwingUtilities.invokeLater(() -> view.showError("Could not add
alignment."));
        }
    }
}

public class AlignmentRemoveListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        selectionBlocked = true;
        // when removing, we get the object from the selection, not the
input text fields!
        view.fillModelFromSelection(model);
        new Thread(this::removeAlignment).start();
    }

    private void removeAlignment() {
        if(alignmentLogic.removeAlignment(model)) {
            SwingUtilities.invokeLater(() -> view.bindModel(model));
        } else {
            SwingUtilities.invokeLater(() -> view.showError("Could not
delete alignment."));
        }
        selectionBlocked = false;
    }
}
}

```

## AlignmentLogic.java

```
package swp4.ue05.part2.logic;

import swp4.ue05.part2.ui.model.AlignmentModel;

public interface AlignmentLogic {
    boolean addAlignment(AlignmentModel model);
    AlignmentModel getAlignments(AlignmentModel model);
    boolean removeAlignment(AlignmentModel model);
}
```

## AlignmentLogicImpl.java

```
package swp4.ue05.part2.logic.impl;

import swp4.ue05.part2.logic.AlignmentLogic;
import swp4.ue05.part2.ui.model.AlignmentModel;

public class AlignmentLogicImpl implements AlignmentLogic {
    @Override
    public boolean addAlignment(AlignmentModel model) {
        // only add an alignment if the model and currentalignment is not null
        return model != null && model.getCurrentAlignment() != null &&
model.addAlignment(model.getCurrentAlignment());
    }

    @Override
    public AlignmentModel getAlignments(AlignmentModel model) {
        return model;
    }

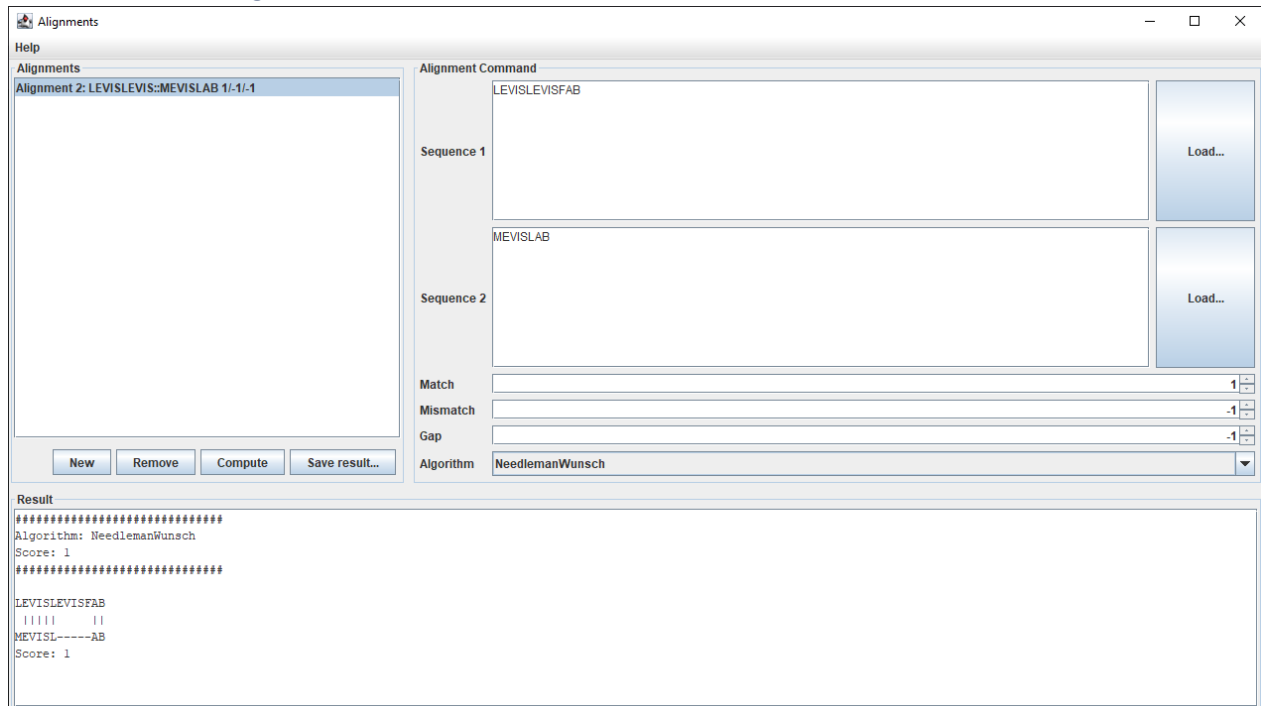
    @Override
    public boolean removeAlignment(AlignmentModel model) {
        // only start removal process if model and currentAlignment is not
null
        return model != null && model.getCurrentAlignment() != null &&
model.removeAlignment(model.getCurrentAlignment());
    }
}
```

## Testfälle

Da die Testfälle schwer zu streamlines sind, wurde dieser Teil nur mittels Beschreibungen und zusätzlichen Screenshots/Ausgaben dokumentiert.

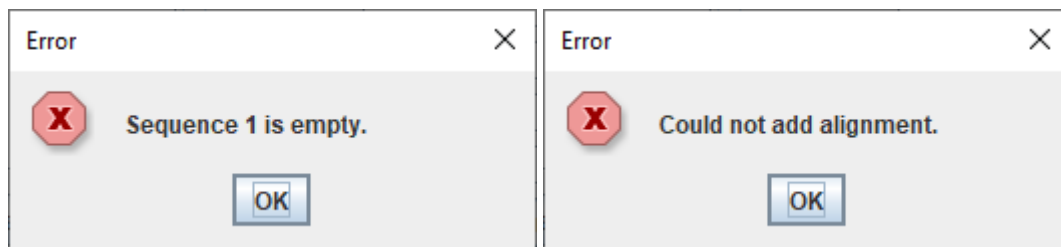
- Normale Benützung
- Leere Sequenzen
- Invalide Sequenzen
- Remove ohne Selection
- Sehr lange Sequenzen

### Normale Benützung



### Leere Sequenzen

Will man eine Alignment-Einstellung hinzufügen, lässt jedoch eine oder mehrere Sequenzen leer, so kommt es zur Warnung über das UI:

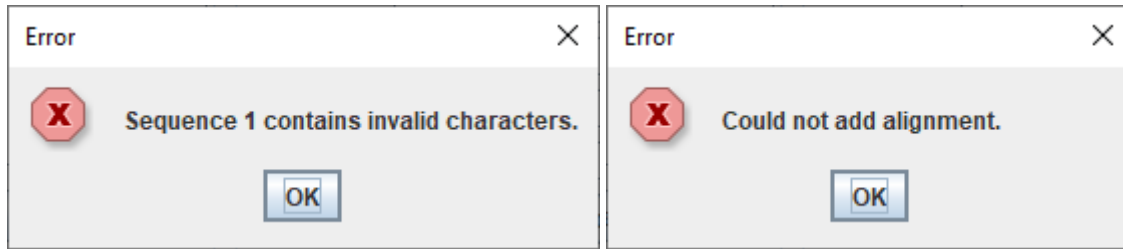


Gleich verhält es sich beim Computen solcher Elemente.

### Invalide Sequenzen

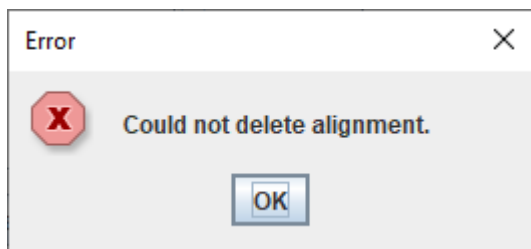
Das Erzeugen eines PairwiseAlignment funktioniert nur mit Buchstaben, weswegen es wichtig ist Sonderzeichen, Zahlen und Whitespaces abzufangen.

Gibt der User „Wie geht's?“ in eines der Sequenzfelder ein, so wird dieser darüber informiert, dass das falsch ist:



#### Remove ohne Selection

Wie beim Personen-GUI ist anfangs kein Eintrag in der Datenquelle ausgewählt. Wird der Remove-Button betätigt, so gibt es keine Alignment-ID und damit kann der Prozess nicht durchgeführt werden.



#### Sehr lange Sequenzen

Bei sehr langen Sequenzen wird der Standardoutput unleserlich. Dadurch, dass die Alignments nicht aufgeteilt werden können, ohne das Ergebnis zu verändern konnte ich es nicht besser darstellen bzw. konnte ich keine Lösung dazu finden.

