```java
package ab;

import static ab.IconUtil.getLargeIcon;
import static ab.IconUtil.getSmallImage;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;

import javax.swing.AbstractAction;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JToolBar;

import ab.AddressDialog.Option;
import ab.table.AddressTable;
import ab.table.AddressTableModel;

/**
 * Visual representation of an address book. This is also the application's main
 * class.
 *
 * @author ahoefer
 *
 */
public class AddressBook extends JFrame {

	private static final long serialVersionUID = 1738422841479223075L;

	private AddressTableModel addressModel = new AddressTableModel();

	private AddressTable table = new AddressTable(addressModel);

	/**
	 * Creates a new address book.
	 */
	public AddressBook() {
		setTitle("SWT 1 - Adressbuch");
		setIconImage(getSmallImage("kit"));
		setDefaultCloseOperation(EXIT_ON_CLOSE);

		Container contentPane = getContentPane();
		contentPane.setLayout(new BorderLayout());

		JScrollPane scrollPane = new JScrollPane();
		scrollPane.setViewportView(table);
		contentPane.add(scrollPane, BorderLayout.CENTER);
		contentPane.add(createControl(), BorderLayout.PAGE_START);

		pack();
		setLocationRelativeTo(null);
	}

	private JToolBar createControl() {
		JToolBar control = new JToolBar();
		control.add(new AddAddressAction());
		control.add(new EditAddressAction());
		control.add(new RemoveAddressAction());
		control.add(new JToolBar.Separator());
		control.add(new ExitAction());
		return control;
	}

	private class AddAddressAction extends AbstractAction {

		private static final long serialVersionUID = 5036490585785108338L;

		public AddAddressAction() {
			super("Hinzufgen");
			putValue(LARGE_ICON_KEY, getLargeIcon("add"));
			putValue(SHORT_DESCRIPTION, "Hinzufgen");
		}

		public void actionPerformed(ActionEvent e) {
			AddressDialog dialog = new AddressDialog(AddressBook.this);
			dialog.setVisible(true);
			if (dialog.getSelectedOption() == Option.SAVE) {
				addressModel.addAddress(dialog.getAddress());
			}
		}
	}
```

```
82
83          private class EditAddressAction extends AbstractAction {
84
85                  private static final long serialVersionUID = -1200406722700964528L;
86
87                  public EditAddressAction() {
88                          super("Bearbeiten");
89                          putValue(LARGE_ICON_KEY, getLargeIcon("edit"));
90                          putValue(SHORT_DESCRIPTION, "Bearbeiten");
91                  }
92
93                  public void actionPerformed(ActionEvent e) {
94                          Address address = addressModel.getAddress(table.getSelectedRow());
95
96                          if (address == null) {
97                                  return;
98                          }
99                          AddressDialog dialog = new AddressDialog(address, AddressBook.this);
100                         dialog.setVisible(true);
101                         if (dialog.getSelectedOption() == Option.SAVE) {
102                                 addressModel.updateAddress(table.getSelectedRow(), dialog
103                                                 .getAddress());
104                         }
105                 }
106         }
107
108         private class RemoveAddressAction extends AbstractAction {
109
110                 private static final long serialVersionUID = -6681795431514148118L;
111
112                 public RemoveAddressAction() {
113                         super("Entfernen");
114                         putValue(LARGE_ICON_KEY, getLargeIcon("remove"));
115                         putValue(SHORT_DESCRIPTION, "Entfernen");
116                 }
117
118                 public void actionPerformed(ActionEvent e) {
119                         addressModel.removeAddress(table.getSelectedRow());
120                 }
121         }
122
123         private class ExitAction extends AbstractAction {
124
125                 private static final long serialVersionUID = 2727566117954481029L;
126
127                 public ExitAction() {
128                         super("Beenden");
129                         putValue(LARGE_ICON_KEY, getLargeIcon("exit"));
130                         putValue(SHORT_DESCRIPTION, "Beenden");
131                 }
132
133                 public void actionPerformed(ActionEvent e) {
134                         System.exit(0);
135                 }
136         }
137
138         /**
139          * Starts the application.
140          *
141          * @param args
142          *            Arguments will be ignored.
143          */
144         public static void main(String[] args) {
145                 AddressBook ab = new AddressBook();
146                 ab.setVisible(true);
147         }
148 }
149
```

Lorenz Diener - 1446555 - Tutorium 7 - Blatt 3 - Aufgabe 4 - AddressTable.java

```java
package ab.table;

import javax.swing.JTable;

/**
 * A table to display the address data.
 *
 * @author ahoefer
 *
 */
public class AddressTable extends JTable {

	/**
	 * Constructor. Needs an AddressTableModel to work with.
	 *
	 * @param model
	 *            The AdressTableModel with the data that needs to be displayed.
	 */
	public AddressTable(AddressTableModel model) {
		super(model);
		this.setDefaultRenderer(Object.class, new AdressTableCellRenderer());
	}

	private static final long serialVersionUID = 1514857297355439104L;
}

```

```java
package ab.table;

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

import javax.swing.table.AbstractTableModel;

import ab.Address;

public class AddressTableModel extends AbstractTableModel {
	private static final long serialVersionUID = -5698713339563803417L;

	private List<Address> adresses = new ArrayList<Address>();

	String columnNames[] = { "Kategorie", "Vorname", "Nachname", "Adresse",
			"PLZ", "Land", "E-Mail", "Telefon", "Mobil" };

	/**
	 * Return the number of columns in this table.
	 *
	 * @return the number of columns in this table.
	 */
	public int getColumnCount() {
		return (columnNames.length);
	}

	/**
	 * Return the number of rows in this table.
	 *
	 * @return the number of rows in this table.
	 */
	public int getRowCount() {
		return (adresses.size());
	}

	/**
	 * Return the name of the column-th column.
	 *
	 * @param The
	 *            number of the column whose name we want.
	 * @return The name of the column'th column.
	 */
	public String getColumnName(int column) {
		return (columnNames[column]);
	}

	/**
	 * Return the value to be displayed at (row, column).
	 *
	 * @param row
	 *            The row where the value you want is.
	 * @param column
	 *            The column where the value you want is.
	 * @return the value to be displayed at (row, column).
	 */
	public Object getValueAt(int row, int column) {
		Address address = this.adresses.get(row);
		switch (column) {
		case 0:
			return (address.getCategory());
		case 1:
			return (address.getFirstname());
		case 2:
			return (address.getLastname());
		case 3:
			return (address.getAddress1() + " " + address.getAddress2() + " " + address
					.getCity());
		case 4:
			return (address.getZipCode());
		case 5:
			return (address.getCounty());
		case 6:
			return (address.getEMail());
		case 7:
			return (address.getLandline());
		case 8:
			return (address.getMobile());
		}
		// If we got here, a wrong column was specified.
		throw new IllegalArgumentException("Wrong column.");
	}
```

```
 82
 83          /**
 84           * Add an address to the table.
 85           *
 86           * @param address
 87           *            The table to add.
 88           */
 89          public void addAddress(Address address) {
 90                  this.adresses.add(new Address(address));
 91                  this.fireTableRowsInserted(this.adresses.size() - 1, this.adresses
 92                                  .size() - 1);
 93          }
 94
 95          /**
 96           * Get an address from the table.
 97           *
 98           * @param addressIndex
 99           *            Which address to get.
100           * @return The address in row row.
101           */
102          public Address getAddress(int addressIndex) {
103                  if (addressIndex < 0 || addressIndex >= this.adresses.size()) {
104                          return (null);
105                  }
106                  return new Address(this.adresses.get(addressIndex));
107          }
108
109          /**
110           * Remove an address from the table.
111           *
112           * @param addressIndex
113           *            The row in which the adress to remove is.
114           */
115          public void removeAddress(int addressIndex) {
116                  if (!(addressIndex < 0 || addressIndex >= this.adresses.size())) {
117                          this.adresses.remove(addressIndex);
118                          this.fireTableRowsDeleted(addressIndex, addressIndex);
119                  }
120          }
121
122          /**
123           * Change a saved address.
124           *
125           * @param addressIndex
126           *            The row of the address you want to change.
127           * @param address
128           *            The new address to save in this row.
129           */
130          public void updateAddress(int addressIndex, Address address) {
131                  this.adresses.set(addressIndex, new Address(address));
132                  this.fireTableRowsUpdated(addressIndex, addressIndex);
133          }
134  }
135
```

```java
package ab.table;

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

import javax.swing.table.AbstractTableModel;

import ab.Address;

public class AddressTableModel extends AbstractTableModel {
	private static final long serialVersionUID = -5698713339563803417L;

	private List<Address> adresses = new ArrayList<Address>();

	String columnNames[] = { "Kategorie", "Vorname", "Nachname", "Adresse",
			"PLZ", "Land", "E-Mail", "Telefon", "Mobil" };

	/**
	 * Return the number of columns in this table.
	 *
	 * @return the number of columns in this table.
	 */
	public int getColumnCount() {
		return (columnNames.length);
	}

	/**
	 * Return the number of rows in this table.
	 *
	 * @return the number of rows in this table.
	 */
	public int getRowCount() {
		return (adresses.size());
	}

	/**
	 * Return the name of the column-th column.
	 *
	 * @param The
	 *            number of the column whose name we want.
	 * @return The name of the column'th column.
	 */
	public String getColumnName(int column) {
		return (columnNames[column]);
	}

	/**
	 * Return the value to be displayed at (row, column).
	 *
	 * @param row
	 *            The row where the value you want is.
	 * @param column
	 *            The column where the value you want is.
	 * @return the value to be displayed at (row, column).
	 */
	public Object getValueAt(int row, int column) {
		Address address = this.adresses.get(row);
		switch (column) {
		case 0:
			return (address.getCategory());
		case 1:
			return (address.getFirstname());
		case 2:
			return (address.getLastname());
		case 3:
			return (address.getAddress1() + " " + address.getAddress2() + " " + address
					.getCity());
		case 4:
			return (address.getZipCode());
		case 5:
			return (address.getCounty());
		case 6:
			return (address.getEMail());
		case 7:
			return (address.getLandline());
		case 8:
			return (address.getMobile());
		}
		// If we got here, a wrong column was specified.
		throw new IllegalArgumentException("Wrong column.");
	}
```

```
 82
 83          /**
 84           * Add an address to the table.
 85           *
 86           * @param address
 87           *            The table to add.
 88           */
 89          public void addAddress(Address address) {
 90                  this.adresses.add(new Address(address));
 91                  this.fireTableRowsInserted(this.adresses.size() - 1, this.adresses
 92                                  .size() - 1);
 93          }
 94
 95          /**
 96           * Get an address from the table.
 97           *
 98           * @param addressIndex
 99           *            Which address to get.
100           * @return The address in row row.
101           */
102          public Address getAddress(int addressIndex) {
103                  if (addressIndex < 0 || addressIndex >= this.adresses.size()) {
104                          return (null);
105                  }
106                  return new Address(this.adresses.get(addressIndex));
107          }
108
109          /**
110           * Remove an address from the table.
111           *
112           * @param addressIndex
113           *            The row in which the adress to remove is.
114           */
115          public void removeAddress(int addressIndex) {
116                  if (!(addressIndex < 0 || addressIndex >= this.adresses.size())) {
117                          this.adresses.remove(addressIndex);
118                          this.fireTableRowsDeleted(addressIndex, addressIndex);
119                  }
120          }
121
122          /**
123           * Change a saved address.
124           *
125           * @param addressIndex
126           *            The row of the address you want to change.
127           * @param address
128           *            The new address to save in this row.
129           */
130          public void updateAddress(int addressIndex, Address address) {
131                  this.adresses.set(addressIndex, new Address(address));
132                  this.fireTableRowsUpdated(addressIndex, addressIndex);
133          }
134  }
135
```