Programowanie w języku JAVA

Projekt etap I i II

Marcin Godfryd grupa 31

**Kod źródłowy**

1. Main.java

import controller.MainController;  
import javax.swing.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(MainController::new);  
 }  
}

1. MainView.java

package view;  
  
import javax.swing.\*;  
  
public class MainView extends JFrame {  
 private JTabbedPane tabbedPane;  
  
 public MainView() {  
 setTitle("System zarządzania zamówieniami");  
 setSize(800, 600);  
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
 setLocationRelativeTo(null);  
  
 tabbedPane = new JTabbedPane();  
 add(tabbedPane);  
 }  
  
 public void addTab(String title, JPanel panel) {  
 tabbedPane.addTab(title, panel);  
 }  
}

1. MainController.java

package controller;  
  
  
import view.customer.CustomerFormView;  
import view.customer.CustomerView;  
import view.MainView;  
import view.order.OrderFormView;  
import view.order.OrderView;  
import view.product.ProductFormView;  
import view.product.ProductView;  
  
public class MainController {  
  
 public MainController() {  
 MainView mainView = new MainView();  
  
 CustomerView customerView = new CustomerView();  
 CustomerFormView customerForm = new CustomerFormView(mainView);  
 CustomerController customerController = new CustomerController(customerView, customerForm);  
 mainView.addTab("Klienci", customerView);  
  
 ProductView productView = new ProductView();  
 ProductFormView productFormView = new ProductFormView(mainView);  
 ProductController productController = new ProductController(productView, productFormView);  
 mainView.addTab("Produkty", productView);  
  
 OrderView orderView = new OrderView();  
 OrderFormView orderFormView = new OrderFormView(mainView);  
 OrderController orderController = new OrderController(orderView, orderFormView);  
 mainView.addTab("Zamówienia", orderView);  
  
 mainView.setVisible(true);  
 }  
  
}

1. AbstractController.java

package controller;  
  
import model.AbstractModel;  
import util.FileUtil;  
import util.ValidatorUtil;  
import view.AbstractFormView;  
import view.AbstractView;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.math.BigDecimal;  
import java.text.ParseException;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Optional;  
  
public abstract class AbstractController<T extends AbstractModel, TView extends AbstractView, TForm extends AbstractFormView> {  
 protected final TView view;  
 protected final TForm form;  
 private final String dataFile;  
 private List<T> dataList;  
 private int currentId = 0;  
 protected boolean isValidate = true;  
 protected StringBuilder errorMsg = new StringBuilder();  
  
 protected AbstractController(TView view, TForm form, String dataFile) {  
 this.view = view;  
 this.form = form;  
 this.dataFile = dataFile;  
 loadFromFile();  
 view.addButtonAction(e -> showForm());  
 view.removeButtonAction(e -> remove());  
 form.submitForm(e -> add());  
 form.cancelForm(e -> cancelForm());  
  
 view.doubleClickAction(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (e.getClickCount() == 2 && e.getButton() == MouseEvent.*BUTTON1*) {  
 Integer selectedRow = view.getSelectedElement();  
 showDetails(findById(selectedRow));  
 }  
 }  
 });  
 }  
  
 protected List<T> getDataList() {  
 return dataList;  
 }  
  
 protected int generateNextId() {  
 int maxId = dataList.stream().mapToInt(T::getId).max().orElse(0);  
 currentId = Math.*max*(currentId, maxId);  
 return currentId + 1;  
 }  
  
 protected T findById(Integer id) {  
 Optional<T> findedElem = dataList.stream().filter(c -> c.getId().equals(id)).findFirst();  
 return findedElem.orElse(null);  
 }  
  
 protected void setSorterToBigDecimalValue(int columnIndex) {  
 view.getSorter().setComparator(columnIndex, new Comparator<BigDecimal>() {  
 @Override  
 public int compare(BigDecimal o1, BigDecimal o2) {  
 if (o1 == null) return (o2 == null) ? 0 : -1;  
 if (o2 == null) return 1;  
 return o1.compareTo(o2);  
 }  
 });  
 }  
  
 protected abstract void showDetails(T element);  
 protected abstract T create();  
 protected abstract void validateFormFields();  
  
 private void saveToFile() {  
 FileUtil.*saveToFile*(dataFile, dataList);  
 }  
  
 private void showForm() {  
 form.setVisible(true);  
 }  
  
 private boolean isValidate() {  
 validateFormFields();  
 if(!errorMsg.isEmpty()) {  
 JOptionPane.*showMessageDialog*(this.form, errorMsg, "Error", JOptionPane.*ERROR\_MESSAGE*);  
 return false;  
 }  
 return true;  
 }  
  
 private void add() {  
 if(isValidate()) {  
 T element = create();  
 dataList.add(element);  
 view.addToView(element);  
 saveToFile();  
 form.clearFormFields();  
 form.setVisible(false);  
 }  
 errorMsg.setLength(0);  
 isValidate = true;  
  
 }  
  
 public void remove() {  
 Integer selectedCustomer = view.getSelectedElement();  
 if (selectedCustomer != null) {  
 dataList.removeIf(c -> c.getId().equals(selectedCustomer));  
 view.removeFromView(selectedCustomer);  
 saveToFile();  
 }  
 }  
  
 private void loadFromFile() {  
 dataList = FileUtil.*loadFromFile*(dataFile);  
 for(T element : dataList) {  
 view.addToView(element);  
 }  
 }  
  
  
 protected void validateAndColorField(JTextField field, String errorMessage) {  
 if (!ValidatorUtil.*validateTextField*(field.getText())) {  
 errorMsg.append(errorMessage);  
 field.setBackground(Color.*PINK*);  
 } else {  
 field.setBackground(Color.*WHITE*);  
 }  
 }  
  
 private void cancelForm() {  
 form.setVisible(false);  
 }  
  
}

1. CustomerController.java

package controller;  
  
import model.Address;  
import model.Customer;  
import model.Order;  
import util.DateTimeUtil;  
import util.FileUtil;  
import util.ValidatorUtil;  
import view.customer.CustomerFormView;  
import view.customer.CustomerView;  
  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.table.TableColumn;  
import javax.swing.table.TableColumnModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.math.BigDecimal;  
import java.util.\*;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class CustomerController extends AbstractController<Customer, CustomerView, CustomerFormView> {  
 private static final String *CUSTOMER\_FILE* = "customers.dat";  
  
 public CustomerController(CustomerView view, CustomerFormView form) {  
 super(view, form, *CUSTOMER\_FILE*);  
 form.showDeliveryPanelYes(e -> setDeliveryAddressFields(true));  
 form.showDeliveryPanelNo(e -> setDeliveryAddressFields(false));  
 view.addActionToFilterButton(e -> search());  
 view.addActionToResetButton(e -> resetFilter());  
 }  
  
  
  
 @Override  
 protected void showDetails(Customer element) {  
 if (element != null) {  
 JTextArea textArea = new JTextArea(10, 30);  
 textArea.setText(createCustomerDetailsText(element));  
 textArea.setEditable(false);  
 textArea.setLineWrap(true);  
 textArea.setWrapStyleWord(true);

textArea.setCaretPosition(0);

JScrollPane scrollPane = new JScrollPane(textArea);  
 JOptionPane.*showMessageDialog*(view, scrollPane, "Szczegóły Klienta", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 }  
  
 private String createCustomerDetailsText(Customer element) {  
 StringBuilder details = new StringBuilder();  
 details.append("Imię: ").append(element.getName()).append("\n");  
 details.append("Nazwisko: ").append(element.getLastname()).append("\n");  
 details.append("Firma: ").append(element.getCompany()).append("\n");  
 details.append("NIP: ").append(element.getNip()).append("\n");  
 details.append("Adres:\n").append(element.getAddress()).append("\n");  
 details.append("Adres dostawy:\n").append(  
 element.getDeliveryAddress() != null ? element.getDeliveryAddress() : "Taki sam jak powyżej"  
 ).append("\n");  
  
 return details.toString();  
 }  
  
 @Override  
 protected Customer create() {  
 Address customerAddress = new Address(  
 form.getStreetField().getText(),  
 form.getHouseNumberField().getText(),  
 form.getApartmentNumberField().getText(),  
 form.getCityField().getText(),  
 form.getPostalCodeField().getText(),  
 form.getStateField().getText(),  
 form.getCountryField().getText()  
 );  
 Address deliveryAddress = null;  
 if (form.getdeliveryAddressYes().isSelected()) {  
 deliveryAddress = new Address(  
 form.getDeliveryStreetField().getText(),  
 form.getDeliveryHouseNumberField().getText(),  
 form.getDeliveryApartmentNumberField().getText(),  
 form.getDeliveryCityField().getText(),  
 form.getDeliveryPostalCodeField().getText(),  
 form.getDeliveryStateField().getText(),  
 form.getDeliveryCountryField().getText()  
 );  
 }  
  
 return new Customer(  
 generateNextId(),  
 form.getNameField().getText(),  
 form.getLastNameField().getText(),  
 form.getCompanyField().getText(),  
 form.getNipField().getText(),  
 customerAddress,  
 deliveryAddress  
 );  
 }  
  
 @Override  
 protected void validateFormFields() {  
 validateAndColorField(form.getNameField(), "Imię jest wymagane.\n");  
 validateAndColorField(form.getLastNameField(), "Nazwisko jest wymagane.\n");  
  
 if (!ValidatorUtil.*validateTextField*(form.getCompanyField().getText()) && !form.getNipField().getText().isEmpty()) {  
 errorMsg.append("Nazwa firmy jest wymagana, gdy podany jest NIP.\n");  
 form.getCompanyField().setBackground(Color.*PINK*);  
 }  
 else {  
 form.getCompanyField().setBackground(Color.*WHITE*);  
 }  
  
 if(!ValidatorUtil.*validateNIP*(form.getNipField().getText())) {  
 errorMsg.append("NIP jest niepoprawny.\n");  
 }  
 else {  
 form.getNipField().setBackground(Color.*WHITE*);  
 }  
  
 if (!ValidatorUtil.*validateTextField*(form.getNipField().getText()) && !form.getCompanyField().getText().isEmpty()) {  
 errorMsg.append("NIP jest wymagany, gdy podana jest nazwa firmy.\n");  
 form.getNipField().setBackground(Color.*PINK*);  
 }  
 else {  
 form.getNipField().setBackground(Color.*WHITE*);  
 }  
  
  
  
 validateAndColorField(form.getStreetField(), "Nazwa ulicy jest wymagana.\n");  
 validateAndColorField(form.getHouseNumberField(), "Numer budynku jest wymagany.\n");  
 validateAndColorField(form.getCityField(), "Miejscowość jest wymagana.\n");  
 validateAndColorField(form.getPostalCodeField(), "Kod pocztowy jest niepoprawny.\n");  
  
 if(!ValidatorUtil.*validatePostalCode*(form.getPostalCodeField().getText())) {  
 errorMsg.append("Kod pocztowy jest niepoprawny.\n");  
 form.getPostalCodeField().setBackground(Color.*PINK*);  
 }  
 else {  
 form.getPostalCodeField().setBackground(Color.*WHITE*);  
 }  
  
 validateAndColorField(form.getStateField(), "Województwo jest wymagane.\n");  
 validateAndColorField(form.getCountryField(), "Kraj jest wymagany.\n");  
  
  
 if (form.getdeliveryAddressYes().isSelected()) {  
 validateAndColorField(form.getDeliveryStreetField(), "Adres dostawy - Nazwa ulicy jest wymagana.\n");  
 validateAndColorField(form.getDeliveryHouseNumberField(), "Adres dostawy - Numer budynku jest wymagany.\n");  
 validateAndColorField(form.getDeliveryCityField(), "Adres dostawy - Miejscowość jest wymagana.\n");  
  
 if(!ValidatorUtil.*validatePostalCode*(form.getDeliveryPostalCodeField().getText())) {  
 errorMsg.append("Adres dostawy - Kod pocztowy jest niepoprawny.\n");  
 form.getDeliveryPostalCodeField().setBackground(Color.*PINK*);  
 }  
 else {  
 form.getDeliveryPostalCodeField().setBackground(Color.*WHITE*);  
 }  
  
 validateAndColorField(form.getDeliveryStateField(), "Adres dostawy - Województwo jest wymagane.\n");  
 validateAndColorField(form.getDeliveryCountryField(), "Adres dostawy - Kraj jest wymagany.\n");  
 }  
 }  
  
 private void setDeliveryAddressFields(boolean enable) {  
 form.setDeliveryAddressFieldsEnabled(enable);  
 if(!enable) {  
 resetDeliveryAddressFieldsColor();  
 }  
 }  
  
 private List<Order> getOrders() {  
 return FileUtil.*loadFromFile*("orders.dat");  
 }  
  
  
 private void search() {  
 String startDateString = view.getStartDateField().getText();  
 String endDateString = view.getEndDateField().getText();  
 Date startDate = DateTimeUtil.*parseDate*(startDateString);  
 Date endDate = DateTimeUtil.*parseDate*(endDateString);  
  
 BigDecimal minOrderValue = null;  
  
 try {  
 if (!view.getMinOrderValueField().getText().isEmpty()) {  
 minOrderValue = new BigDecimal(view.getMinOrderValueField().getText());  
 }  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(null, "Wprowadzono nieprawidłową wartość. Proszę wprowadzić liczbę.");  
 return;  
 }  
 boolean startFilter = (startDate != null || endDate != null || minOrderValue != null);  
 if (!startFilter) {  
 return ;  
 }  
  
 Map<Customer, BigDecimal> customerOrderTotalMap = getDataList().stream()  
 .collect(Collectors.*toMap*(  
 customer -> customer,  
 customer -> BigDecimal.*ZERO* ));  
  
 Map<Customer, BigDecimal> ordersMap = getOrders().stream()  
 .filter(order -> (startDate == null || !order.getDate().before(startDate)) &&  
 (endDate == null || !order.getDate().after(endDate)))  
 .collect(Collectors.*groupingBy*(  
 Order::getClient,  
 Collectors.*reducing*(  
 BigDecimal.*ZERO*,  
 Order::getOrderTotalPrice,  
 BigDecimal::add  
 )  
 ));  
  
 ordersMap.forEach((customer, totalValue) ->  
 customerOrderTotalMap.merge(customer, totalValue, BigDecimal::add));  
  
 BigDecimal finalMinOrderValue = minOrderValue;  
 Set<Customer> customersMeetingCriteria = ordersMap.entrySet().stream()  
 .filter(entry -> entry.getValue().compareTo(BigDecimal.*ZERO*) > 0) // Dodatkowe sprawdzenie, czy klient złożył jakiekolwiek zamówienia  
 .filter(entry -> (finalMinOrderValue == null || entry.getValue().compareTo(finalMinOrderValue) >= 0))  
 .map(Map.Entry::getKey)  
 .collect(Collectors.*toSet*());  
  
 TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>((DefaultTableModel) view.getTable().getModel());  
 view.getTable().setRowSorter(sorter);  
  
 sorter.setRowFilter(new RowFilter<DefaultTableModel, Integer>() {  
 @Override  
 public boolean include(Entry<? extends DefaultTableModel, ? extends Integer> entry) {  
  
 String clientId = entry.getStringValue(0);  
 Customer clientInRow = findById(Integer.*parseInt*(clientId));  
 updateTableWithOrderSum(customerOrderTotalMap);  
  
 return customersMeetingCriteria.contains(clientInRow);  
 }  
 });  
 }  
  
 private void updateTableWithOrderSum(Map<Customer, BigDecimal> customerMap) {  
 DefaultTableModel model = (DefaultTableModel) view.getTable().getModel();  
  
 if (model.findColumn("Suma Zamówień") == -1) {  
 model.addColumn("Suma Zamówień");  
 }  
  
 for (int i = 0; i < model.getRowCount(); i++) {  
 Integer clientId = Integer.*parseInt*(model.getValueAt(i, 0).toString());  
  
 Customer customer = findById(clientId);  
 BigDecimal orderSum = customerMap.getOrDefault(customer, BigDecimal.*ZERO*);  
  
 model.setValueAt(orderSum, i, model.getColumnCount() - 1);  
 }  
 int newColumnIndex = model.getColumnCount() - 1;  
 setSorterToBigDecimalValue(newColumnIndex);  
 }  
  
 private void removeOrderSumColumn() {  
 DefaultTableModel model = (DefaultTableModel) view.getTable().getModel();  
  
 int orderSumColumnIndex = model.findColumn("Suma Zamówień");  
  
 if (orderSumColumnIndex != -1) {  
 TableColumnModel columnModel = view.getTable().getColumnModel();  
 TableColumn orderSumColumn = columnModel.getColumn(orderSumColumnIndex);  
 view.getTable().removeColumn(orderSumColumn);  
 model.setColumnCount(model.getColumnCount() - 1);  
 }  
 }  
  
 private void resetFilter() {  
 TableRowSorter<DefaultTableModel> sorter = (TableRowSorter<DefaultTableModel>) view.getTable().getRowSorter();  
 sorter.setRowFilter(null);  
 view.getTable().clearSelection();  
 view.getStartDateField().setText("");  
 view.getEndDateField().setText("");  
 view.getMinOrderValueField().setText("");  
 removeOrderSumColumn();  
 }  
  
 private void resetDeliveryAddressFieldsColor() {  
 form.getDeliveryStreetField().setBackground(Color.*WHITE*);  
 form.getDeliveryHouseNumberField().setBackground(Color.*WHITE*);  
 form.getDeliveryApartmentNumberField().setBackground(Color.*WHITE*);  
 form.getDeliveryCityField().setBackground(Color.*WHITE*);  
 form.getDeliveryPostalCodeField().setBackground(Color.*WHITE*);  
 form.getDeliveryStateField().setBackground(Color.*WHITE*);  
 form.getDeliveryCountryField().setBackground(Color.*WHITE*);  
 }  
  
}

1. OrderController.java

package controller;  
  
import model.\*;  
import util.DateTimeUtil;  
import util.FileUtil;  
import util.ValidatorUtil;  
import view.order.OrderFormView;  
import view.order.OrderView;  
  
import javax.swing.\*;  
import javax.swing.event.PopupMenuEvent;  
import javax.swing.event.PopupMenuListener;  
import javax.swing.event.TableModelEvent;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.math.BigDecimal;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
import java.util.List;  
  
public class OrderController extends AbstractController<Order, OrderView, OrderFormView> {  
  
 private Customer selectedCustomer;  
 private Customer latestCustomer;  
 private Date orderDate;  
 private int quantity;  
 private int discount;  
 private List<ItemsList> itemsList = new ArrayList<>();  
  
 protected OrderController(OrderView view, OrderFormView orderFormView) {  
 super(view, orderFormView, "orders.dat");  
 form.addActionToRemoveButton(e -> removeProductFromOrderList());  
 form.addActionToSelectButton(e -> addProductsToTable());  
 form.addActionToProductTableModel(this::calculateTotalSum);  
 form.addActionToAddButton(e -> updateAvailableProducts());  
 form.addActionToCustomerComboBox(new PopupMenuListener() {  
 public void popupMenuWillBecomeVisible(PopupMenuEvent e) {  
 addCustomersToFormComboBox(form.getCustomerComboBox());  
 }  
  
 public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {  
 updateDeliveryAddressFields();  
 }  
  
 public void popupMenuCanceled(PopupMenuEvent e) {}  
 });  
  
 view.addActionToCustomerComboBox(new PopupMenuListener() {  
 public void popupMenuWillBecomeVisible(PopupMenuEvent e) {  
 addCustomersToComboBox(view.getCustomerComboBox());  
 }  
  
 public void popupMenuWillBecomeInvisible(PopupMenuEvent e) {  
 }  
  
 public void popupMenuCanceled(PopupMenuEvent e) {}  
 });  
  
 view.addActionToFilterButton(e -> search());  
 view.addActionToResetButton(e -> resetFilter());  
  
 form.getProductComboBox().setRenderer(new DefaultListCellRenderer() {  
 @Override  
 public Component getListCellRendererComponent(JList<?> list, Object value, int index, boolean isSelected, boolean cellHasFocus) {  
 super.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);  
  
 if (value instanceof Product) {  
 Product product = (Product) value;  
 setText(product.getName());  
 }  
  
 return this;  
 }  
 });  
 setSorterToBigDecimalValue(2);  
 }  
  
  
 @Override  
 protected void showDetails(Order element) {  
 if (element != null) {  
 JTextArea textArea = new JTextArea(15, 50);  
 textArea.setText(createOrderDetailsText(element));  
 textArea.setEditable(false);  
 textArea.setLineWrap(true);  
 textArea.setWrapStyleWord(true);

textArea.setCaretPosition(0);

JScrollPane scrollPane = new JScrollPane(textArea);  
 JOptionPane.*showMessageDialog*(view, scrollPane, "Szczegóły zamówienia", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 }  
  
 private String createOrderDetailsText(Order element) {  
 StringBuilder details = new StringBuilder();  
  
 details.append("Identyfikator: \n").append(element.getId()).append("\n\n");  
 details.append("Data: \n").append(DateTimeUtil.*showDate*(element.getDate())).append("\n\n");  
 details.append("Klient: \n").append(element.getClient()).append("\n\n");  
 details.append("Produkty: \n");  
 details.append(String.*format*("%-10s %-30s %-10s %-10s %-15s %-15s\n", "ID", "Nazwa", "Ilość", "Rabat", "Netto", "Brutto"));  
 details.append("----------------------------------------------------------------------------------------\n");  
 for (ItemsList item : element.getItemsList()) {  
 details.append(String.*format*("%-10d %-30s %-10d %-10d %-15.2f %-15.2f\n",  
 item.getId(),  
 item.getName(),  
 item.getQuantity(),  
 item.getDiscount(),  
 item.getNetTotal(),  
 item.getGrossTotal()));  
 }  
 details.append("\n");  
 details.append("Cena całkowita: \n").append(element.getOrderTotalPrice()).append(" zł\n\n");  
 details.append("Adres dostawy:\n").append(element.getDeliveryAddress()).append("\n\n");  
  
 return details.toString();  
 }  
  
 @Override  
 protected Order create() {  
 List<ItemsList> orderItems = new ArrayList<>(itemsList);  
  
 return new Order(  
 generateNextId(),  
 orderDate,  
 orderItems,  
 selectedCustomer,  
 createAddress()  
 );  
 }  
  
  
  
  
 @Override  
 protected void validateFormFields() {  
 try {  
 SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");  
 sdf.setLenient(false);  
 orderDate = sdf.parse(form.getOrderDateField().getText());  
 form.getOrderDateField().setBackground(Color.*white*);  
 } catch (Exception e) {  
 errorMsg.append("Data powinna być zapisana w formacie dd-MM-yyyy.\n");  
 form.getOrderDateField().setBackground(Color.*pink*);  
 }  
  
 if(selectedCustomer == null) {  
 errorMsg.append("Nie wybrano klienta.\n");  
 form.getCustomerComboBox().setBackground(Color.*pink*);  
 }  
 else {  
 form.getCustomerComboBox().setBackground(Color.*white*);  
 }  
  
 validateAndColorField(form.getDeliveryStreetField(), "Adres dostawy - nazwa ulicy jest wymagana.\n");  
 validateAndColorField(form.getDeliveryHouseNumberField(), "Adres dostawy - numer budynku jest wymagany.\n");  
 validateAndColorField(form.getDeliveryCityField(), "Adres dostawy - miejscowość jest wymagana.\n");  
 validateAndColorField(form.getDeliveryStateField(), "Adres dostawy - województwo jest wymagane.\n");  
 validateAndColorField(form.getDeliveryCountryField(), "Adres dostawy - kraj jest wymagany.\n");  
  
 if(!ValidatorUtil.*validatePostalCode*(form.getDeliveryPostalCodeField().getText())) {  
 errorMsg.append("Adres dostawy - Kod pocztowy jest niepoprawny.\n");  
 form.getDeliveryPostalCodeField().setBackground(Color.*PINK*);  
 }  
 else {  
 form.getDeliveryPostalCodeField().setBackground(Color.*WHITE*);  
 }  
  
 validateAndProcessProductTable();  
 }  
  
 private Address createAddress() {  
 return new Address(  
 form.getDeliveryStreetField().getText(),  
 form.getDeliveryHouseNumberField().getText(),  
 form.getDeliveryApartmentNumberField().getText(),  
 form.getDeliveryCityField().getText(),  
 form.getDeliveryPostalCodeField().getText(),  
 form.getDeliveryStateField().getText(),  
 form.getDeliveryCountryField().getText()  
 );  
 }  
  
 private void validateAndProcessProductTable() {  
 DefaultTableModel tableModel = form.getproductTableModel();  
 itemsList.clear();  
  
 if (tableModel.getRowCount() == 0) {  
 errorMsg.append("Lista produktów jest pusta. Dodaj przynajmniej jeden produkt.\n");  
 return;  
 }  
  
 for (int row = 0; row < tableModel.getRowCount(); row++) {  
 try {  
 Integer productId = (Integer) tableModel.getValueAt(row, 0);;  
 String productName = (String) tableModel.getValueAt(row, 1);  
 BigDecimal nettoPrice = new BigDecimal(tableModel.getValueAt(row, 2).toString());  
 BigDecimal grossPrice = new BigDecimal(tableModel.getValueAt(row, 3).toString());  
 quantity = Integer.*parseInt*(tableModel.getValueAt(row, 4).toString());  
 discount = Integer.*parseInt*(tableModel.getValueAt(row, 5).toString());  
  
 if (quantity <= 0) {  
 errorMsg.append("Ilość musi być większa od 0: " + productName + ".\n");  
 continue;  
 }  
  
 if (discount < 0 || discount > 100) {  
 errorMsg.append("Rabat musi być większy od 0: " + productName + ".\n");  
 }  
  
 addProductToItemList(productId, quantity, productName, discount, calculateTotal(nettoPrice, quantity), calculateTotal(grossPrice, quantity));  
  
 } catch (NumberFormatException e) {  
 errorMsg.append("Wartości muszą być liczbami dla produktu: " + tableModel.getValueAt(row, 1) + ".\n");  
 } catch (NullPointerException e) {  
 errorMsg.append("Wszystkie pola muszą być wypełnione dla produktu: " + tableModel.getValueAt(row, 1) + ".\n");  
 }  
 }  
 }  
  
 private void addProductToItemList(Integer id, int quantity, String name, int discount, BigDecimal netTotal, BigDecimal grossTotal) {  
 itemsList.add(new ItemsList(id, quantity, name, discount, netTotal, grossTotal));  
 }  
  
 private List<Customer> getCustomers() {  
 return FileUtil.*loadFromFile*("customers.dat");  
 }  
  
 private List<Product> getProducts() {  
 return FileUtil.*loadFromFile*("products.dat");  
 }  
  
 private void addCustomersToFormComboBox(JComboBox<Customer> comboBox) {  
 addCustomersToComboBox(comboBox);  
  
 if (latestCustomer != null) {  
 for (int i = 0; i < comboBox.getItemCount(); i++) {  
 if (comboBox.getItemAt(i).equals(latestCustomer)) {  
 comboBox.setSelectedIndex(i);  
 break;  
 }  
 }  
 }  
 }  
  
 private void addCustomersToComboBox(JComboBox<Customer> comboBox) {  
 comboBox.removeAllItems();  
 List<Customer> dataList = getCustomers();  
 for(Customer customer : dataList) {  
 comboBox.addItem(customer);  
 }  
 comboBox.setSelectedIndex(-1);  
 }  
  
 private void updateDeliveryAddressFields() {  
 if (!form.getCustomerComboBox().isFocusOwner()) {  
 return;  
 }  
  
 selectedCustomer = (Customer) form.getCustomerComboBox().getSelectedItem();  
  
 if (selectedCustomer != null) {  
 latestCustomer = selectedCustomer;  
 Address addr = null;  
 if(selectedCustomer.getDeliveryAddress() != null) {  
 addr = selectedCustomer.getDeliveryAddress();  
 }  
 else {  
 addr = selectedCustomer.getAddress();  
 }  
  
 form.getDeliveryStreetField().setText(addr.getStreet());  
 form.getDeliveryHouseNumberField().setText(addr.getHouseNumber());  
 form.getDeliveryApartmentNumberField().setText(addr.getApartmentNumber());  
 form.getDeliveryCityField().setText(addr.getCity());  
 form.getDeliveryPostalCodeField().setText(addr.getPostalCode());  
 form.getDeliveryStateField().setText(addr.getState());  
 form.getDeliveryCountryField().setText(addr.getCountry());  
 }  
 }  
  
 private Integer getSelectedElementFromProductList() {  
 int selectedRow = form.getProductTable().getSelectedRow();  
 if (selectedRow >= 0) {  
 return (Integer) form.getproductTableModel().getValueAt(selectedRow, 0);  
 }  
 return null;  
 }  
  
 private void addProductsToTable() {  
 Product selectedProduct = (Product) form.getProductComboBox().getSelectedItem();  
 if (selectedProduct != null) {  
 Object[] rowData = new Object[] {  
 selectedProduct.getId(),  
 selectedProduct.getName(),  
 selectedProduct.getNetPrice(),  
 selectedProduct.getGrossPrice(),  
 1, // default qty  
 0, // default discount  
 selectedProduct.getGrossPrice(),  
 };  
 form.getproductTableModel().addRow(rowData);  
 form.getProductComboBox().removeItem(selectedProduct);  
 }  
 }  
  
 private void updateAvailableProducts() {  
 List<Product> latestProducts = getProducts();  
  
 Set<Integer> productIdsInOrder = new HashSet<>();  
 for (int i = 0; i < form.getproductTableModel().getRowCount(); i++) {  
 Integer productId = (Integer) form.getproductTableModel().getValueAt(i, 0);  
 productIdsInOrder.add(productId);  
 }  
  
 form.getProductComboBox().removeAllItems();  
  
 for (Product product : latestProducts) {  
 if (!productIdsInOrder.contains(product.getId())) {  
 form.getProductComboBox().addItem(product);  
 }  
 }  
 }  
  
 private BigDecimal calculateTotal(BigDecimal price, int quantity) {  
 return price.multiply(BigDecimal.*valueOf*(quantity));  
 }  
  
 private void calculateTotalSum(TableModelEvent e) {  
 int row = e.getFirstRow();  
 int column = e.getColumn();  
  
 if (column == 4 || column == 5) {  
 BigDecimal cenaBrutto = null;  
 int quantity = 0;  
 int discount = 0;  
  
 try {  
 cenaBrutto = new BigDecimal(form.getproductTableModel().getValueAt(row, 3).toString());  
 quantity = Integer.*parseInt*(form.getproductTableModel().getValueAt(row, 4).toString());  
  
 Object rabatObj = form.getproductTableModel().getValueAt(row, 5);  
 if (rabatObj != null) {  
 discount = Integer.*parseInt*(rabatObj.toString());  
  
 if (discount < 0 || discount > 100) {  
 JOptionPane.*showMessageDialog*(null, "Rabat musi być wartością od 0 do 100.");  
 return;  
 }  
 }  
  
 if (quantity <= 0) {  
 JOptionPane.*showMessageDialog*(null, "Ilość musi być większa od 0.");  
 return;  
 }  
  
 BigDecimal rabatDecimal = BigDecimal.*valueOf*(discount).divide(BigDecimal.*valueOf*(100));  
 BigDecimal newGrossPrice = cenaBrutto.multiply(BigDecimal.*valueOf*(quantity)).multiply(BigDecimal.*ONE*.subtract(rabatDecimal));  
  
 form.getproductTableModel().setValueAt(newGrossPrice.setScale(2, BigDecimal.*ROUND\_HALF\_UP*), row, 6);  
 } catch (NumberFormatException exception) {  
 JOptionPane.*showMessageDialog*(null, "Podane wartości muszą być liczbami.");  
 } catch (NullPointerException exception) {  
 JOptionPane.*showMessageDialog*(null, "Wszystkie pola muszą być wypełnione.");  
 }  
 }  
 }  
  
 private void removeProductFromOrderList() {  
 Integer selectedElement = getSelectedElementFromProductList();  
 if (selectedElement != null) {  
 for (int i = 0; i < form.getproductTableModel().getRowCount(); i++) {  
 if (form.getproductTableModel().getValueAt(i, 0).equals(selectedElement)) {  
 form.getproductTableModel().removeRow(i);  
 updateAvailableProducts();  
 break;  
 }  
 }  
 }  
 }  
  
 private void search() {  
 String startDateString = view.getStartDateField().getText();  
 String endDateString = view.getEndDateField().getText();  
  
 Date startDate = DateTimeUtil.*parseDate*(startDateString);  
 Date endDate = DateTimeUtil.*parseDate*(endDateString);  
  
 Customer selectedCustomer = (Customer) view.getCustomerComboBox().getSelectedItem();  
 BigDecimal minOrderValue = null;  
 BigDecimal maxOrderValue = null;  
  
 try {  
 if (!view.getMinOrderValueField().getText().isEmpty()) {  
 minOrderValue = new BigDecimal(view.getMinOrderValueField().getText());  
 }  
 if (!view.getMaxOrderValueField().getText().isEmpty()) {  
 maxOrderValue = new BigDecimal(view.getMaxOrderValueField().getText());  
 }  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(null, "Wprowadzono nieprawidłową wartość. Proszę wprowadzić liczbę.");  
 return;  
 }  
  
 TableRowSorter<DefaultTableModel> sorter = (TableRowSorter<DefaultTableModel>) view.getTable().getRowSorter();  
 BigDecimal finalMaxOrderValue = maxOrderValue;  
 BigDecimal finalMinOrderValue = minOrderValue;  
 sorter.setRowFilter(new RowFilter<>() {  
 @Override  
 public boolean include(Entry<? extends DefaultTableModel, ? extends Integer> entry) {  
 boolean dateInRange = true;  
 if (startDate != null && endDate != null) {  
 try {  
 Date orderDate = view.getDataFormatter().parse(entry.getStringValue(1));  
 dateInRange = !orderDate.before(startDate) && !orderDate.after(endDate);  
 } catch (ParseException ex) {  
 return false;  
 }  
 }  
  
 String orderId = entry.getStringValue(0);  
 Order foundOrder = findById(Integer.*parseInt*(orderId));  
 boolean customerMatches = (selectedCustomer == null || foundOrder.getClient().equals(selectedCustomer));  
  
 boolean orderValueMatches = true;  
 if (finalMinOrderValue != null || finalMaxOrderValue != null) {  
 BigDecimal orderValue = new BigDecimal(entry.getStringValue(2));  
 if (finalMinOrderValue != null && orderValue.compareTo(finalMinOrderValue) < 0) {  
 orderValueMatches = false;  
 }  
 if (finalMaxOrderValue != null && orderValue.compareTo(finalMaxOrderValue) > 0) {  
 orderValueMatches = false;  
 }  
 }  
  
 return dateInRange && customerMatches && orderValueMatches;  
 }  
 });  
 }  
  
 private void resetFilter() {  
 TableRowSorter<DefaultTableModel> sorter = (TableRowSorter<DefaultTableModel>) view.getTable().getRowSorter();  
 sorter.setRowFilter(null);  
 view.getTable().clearSelection();  
 view.getStartDateField().setText("");  
 view.getEndDateField().setText("");  
 view.getMinOrderValueField().setText("");  
 view.getMaxOrderValueField().setText("");  
 }  
}

1. ProductController.java

package controller;  
  
import model.Dimensions;  
import model.Product;  
import util.ValidatorUtil;  
import view.product.ProductFormView;  
import view.product.ProductView;  
  
import javax.swing.\*;  
import javax.swing.event.DocumentEvent;  
import javax.swing.event.DocumentListener;  
import java.awt.\*;  
import java.math.BigDecimal;  
import java.math.RoundingMode;  
  
public class ProductController extends AbstractController <Product, ProductView, ProductFormView>{  
  
 public ProductController(ProductView view, ProductFormView form) {  
 super(view, form, "products.dat");  
  
 view.searchAction(new DocumentListener() {  
 public void updateSearch(DocumentEvent e) {  
 search(view.getFilterField().getText());  
 }  
  
 @Override  
 public void insertUpdate(DocumentEvent e) {  
 updateSearch(e);  
 }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) {  
 updateSearch(e);  
 }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) {  
 updateSearch(e);  
 }  
 });  
 setSorterToBigDecimalValue(3);  
 setSorterToBigDecimalValue(4);  
 }  
  
  
  
 @Override  
 protected void showDetails(Product element) {  
 if (element != null) {  
 JTextArea textArea = new JTextArea(15, 50);  
 textArea.setText(createProductDetailsText(element));  
 textArea.setEditable(false);  
 textArea.setLineWrap(true);  
 textArea.setWrapStyleWord(true);  
 textArea.setCaretPosition(0);  
  
 JScrollPane scrollPane = new JScrollPane(textArea);  
 JOptionPane.*showMessageDialog*(view, scrollPane, "Szczegóły produktu", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 }  
  
 private String createProductDetailsText(Product element) {  
 StringBuilder details = new StringBuilder();  
 details.append("Nazwa: ").append(element.getName()).append("\n\n");  
 details.append("Opis: \n").append(element.getDescription()).append("\n\n");  
 details.append("SKU: ").append(element.getSku()).append("\n\n");  
 details.append("Netto: ").append(element.getNetPrice()).append(" zł\n\n");  
 details.append("Podatek: ").append(element.getTax()).append("%\n\n");  
 details.append("Brutto:\n").append(element.getGrossPrice()).append(" zł\n\n");  
  
 details.append("Waga: ");  
 if (element.getWeight() != null) {  
 details.append(element.getWeight()).append(" kg\n\n");  
 } else {  
 details.append("Brak\n\n");  
 }  
  
 details.append("Wymiary:\n");  
 if (element.getDimensions() != null) {  
 details.append("Długość: ").append(element.getDimensions().getLength()).append(" cm\n")  
 .append("Szerokość: ").append(element.getDimensions().getWidth()).append(" cm\n")  
 .append("Wysokość: ").append(element.getDimensions().getHeight()).append(" cm\n");  
 } else {  
 details.append("Brak\n");  
 }  
  
 return details.toString();  
 }  
  
 @Override  
 protected Product create() {  
 BigDecimal netPrice = new BigDecimal(form.getNetPriceField().getText());  
 int tax = Integer.*parseInt*(form.getTaxField().getText());  
 BigDecimal grossPrice = netPrice.add(  
 netPrice.multiply(BigDecimal.*valueOf*(tax).divide(new BigDecimal("100")))  
 );  
 grossPrice = grossPrice.setScale(2, RoundingMode.*HALF\_UP*);  
  
 Dimensions dimensions = createDimensions(form.getLengthField().getText(), form.getWidthField().getText(), form.getHeightField().getText());  
  
 return new Product(  
 generateNextId(),  
 form.getNameField().getText(),  
 form.getDescriptionArea().getText(),  
 form.getSkuField().getText(),  
 netPrice,  
 grossPrice,  
 tax,  
 dimensions,  
 ValidatorUtil.*validateTextField*(form.getWeightField().getText()) ? Double.*parseDouble*(form.getWeightField().getText()) : null  
  
 );  
 }  
  
  
  
 @Override  
 protected void validateFormFields() {  
  
 validateAndColorField(form.getNameField(), "Nazwa produktu jest wymagana.\n");  
 validateAndColorField(form.getSkuField(), "SKU jest wymagane.\n");  
  
 validatePriceField();  
 validateTaxField();  
  
 int dimensionsValidationResult = ValidatorUtil.*validateDimensions*(form.getWidthField().getText(), form.getHeightField().getText(), form.getLengthField().getText());  
 validateDimensionsFields(dimensionsValidationResult);  
  
 int weightValidationResult = ValidatorUtil.*validateWeight*(form.getWeightField().getText());  
 validateWeightField(weightValidationResult);  
 }  
  
 private void search(String str) {  
 if (str.isEmpty()) {  
 view.getSorter().setRowFilter(null);  
 } else {  
 view.getSorter().setRowFilter(RowFilter.*regexFilter*("(?i)" + str, 1));  
 }  
 }  
  
 private Dimensions createDimensions(String lengthStr, String widthStr, String heightStr) {  
 if (!lengthStr.isEmpty() && !widthStr.isEmpty() && !heightStr.isEmpty()) {  
 double length = Double.*parseDouble*(lengthStr);  
 double width = Double.*parseDouble*(widthStr);  
 double height = Double.*parseDouble*(heightStr);  
 if (length > 0 && width > 0 && height > 0) {  
 return new Dimensions(length, width, height);  
 }  
 }  
 return null;  
 }  
  
 private void validatePriceField() {  
 try {  
 BigDecimal price = new BigDecimal(form.getNetPriceField().getText());  
 if (!ValidatorUtil.*validatePrice*(price)) {  
 errorMsg.append("Cena musi być większa niż 0 i nie może mieć więcej niż dwie cyfry po przecinku.\n");  
 form.getNetPriceField().setBackground(Color.*PINK*);  
 } else {  
 form.getNetPriceField().setBackground(Color.*WHITE*);  
 }  
 } catch (NumberFormatException e) {  
 errorMsg.append("Cena musi być liczbą.\n");  
 form.getNetPriceField().setBackground(Color.*PINK*);  
 }  
 }  
  
 private void validateTaxField() {  
 try {  
 int tax = Integer.*parseInt*(form.getTaxField().getText());  
 if (!ValidatorUtil.*validateIntRange*(tax, 0, 100)) {  
 errorMsg.append("Podatek powinien być z zakresu od 0 do 100.\n");  
 form.getTaxField().setBackground(Color.*PINK*);  
 } else {  
 form.getTaxField().setBackground(Color.*WHITE*);  
 }  
 } catch (NumberFormatException e) {  
 errorMsg.append("Podatek powinien być liczbą.\n");  
 form.getTaxField().setBackground(Color.*PINK*);  
 }  
 }  
  
 private void validateDimensionsFields(int validationResult) {  
 switch (validationResult) {  
 case 0:  
 errorMsg.append("Wymiar jest mniejszy od 0.\n");  
 form.getWidthField().setBackground(Color.*PINK*);  
 form.getHeightField().setBackground(Color.*PINK*);  
 form.getLengthField().setBackground(Color.*PINK*);  
 break;  
 case 1:  
 errorMsg.append("Nie wszystkie wymiary są podane.\n");  
 form.getWidthField().setBackground(Color.*PINK*);  
 form.getHeightField().setBackground(Color.*PINK*);  
 form.getLengthField().setBackground(Color.*PINK*);  
 break;  
 case 2:  
 errorMsg.append("Wymiary powinny być liczbą całkowitą.\n");  
 form.getWidthField().setBackground(Color.*PINK*);  
 form.getHeightField().setBackground(Color.*PINK*);  
 form.getLengthField().setBackground(Color.*PINK*);  
 break;  
 default:  
 form.getWidthField().setBackground(Color.*WHITE*);  
 form.getHeightField().setBackground(Color.*WHITE*);  
 form.getLengthField().setBackground(Color.*WHITE*);  
 }  
 }  
  
 private void validateWeightField(int validationResult) {  
 switch (validationResult) {  
 case 1:  
 errorMsg.append("Waga jest mniejsza lub równa 0.\n");  
 form.getWeightField().setBackground(Color.*PINK*);  
 break;  
 case 2:  
 errorMsg.append("Waga nie jest liczbą.\n");  
 form.getWeightField().setBackground(Color.*PINK*);  
 break;  
 default:  
 form.getWeightField().setBackground(Color.*WHITE*);  
 }  
 }  
}

1. AbstractModel.java

package model;  
  
import java.io.Serializable;  
  
public class AbstractModel implements Serializable {  
 protected Integer id;  
 private static final long *serialVersionUID* = 1L;  
  
 public AbstractModel(Integer id) {  
 this.id = id;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
}

1. Address.java

package model;  
  
import java.io.Serializable;  
import java.util.Objects;  
  
public class Address implements Serializable {  
 private String street;  
 private String houseNumber;  
 private String apartmentNumber; // optional  
 private String city;  
 private String postalCode;  
 private String state;  
 private String country;  
  
 public Address(String street, String houseNumber, String apartmentNumber, String city, String postalCode, String state, String country) {  
 this.street = street;  
 this.houseNumber = houseNumber;  
 this.apartmentNumber = apartmentNumber;  
 this.city = city;  
 this.postalCode = postalCode;  
 this.state = state;  
 this.country = country;  
 }  
  
 public String getStreet() {  
 return street;  
 }  
  
 public String getHouseNumber() {  
 return houseNumber;  
 }  
  
 public String getApartmentNumber() {  
 return apartmentNumber;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public String getPostalCode() {  
 return postalCode;  
 }  
  
 public String getState() {  
 return state;  
 }  
  
 public String getCountry() {  
 return country;  
 }  
 @Override  
 public String toString() {  
 return "Ulica: " + street + "\n\n" +  
 "Numer budynku: " + houseNumber + "\n\n" +  
 "Numer mieszkania: " + (apartmentNumber.isEmpty() ? "N/A" : apartmentNumber) + "\n\n" +  
 "Miasto: " + city + "\n\n" +  
 "Kod pocztowy: " + postalCode + "\n\n" +  
 "Województwo: " + state + "\n\n" +  
 "Kraj: " + country + "\n\n";  
 }  
  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Address address = (Address) o;  
 return Objects.equals(street, address.street) &&  
 Objects.equals(houseNumber, address.houseNumber) &&  
 Objects.equals(apartmentNumber, address.apartmentNumber) &&  
 Objects.equals(city, address.city) &&  
 Objects.equals(postalCode, address.postalCode) &&  
 Objects.equals(state, address.state) &&  
 Objects.equals(country, address.country);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.hash(street, houseNumber, apartmentNumber, city, postalCode, state, country);  
 }  
}

1. Customer.java

package model;  
  
  
import java.util.Objects;  
  
public class Customer extends AbstractModel {  
 private String name;  
 private String lastname;  
 private String company; // optional  
 private String nip; // optional  
 private Address address;  
 private Address deliveryAddress; // optional  
  
 public Customer(Integer id, String name, String lastname, String company, String nip, Address address, Address deliveryAddress) {  
 super(id);  
 this.name = name;  
 this.lastname = lastname;  
 this.company = company;  
 this.nip = nip;  
 this.address = address;  
 this.deliveryAddress = deliveryAddress;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getLastname() {  
 return lastname;  
 }  
  
 public String getCompany() {  
 return company;  
 }  
  
 public String getNip() {  
 return nip;  
 }  
  
 public Address getAddress() {  
 return address;  
 }  
  
 public Address getDeliveryAddress() {  
 return deliveryAddress;  
 }  
  
  
 @Override  
 public String toString() {  
 return name + " " + lastname + " " + company + " " + nip;  
 }  
  
 public boolean equals(Object o) {  
 if (this == o) return true;  
  
 if (o == null || getClass() != o.getClass()) return false;  
  
 Customer customer = (Customer) o;  
  
 return Objects.equals(getId(), customer.getId()) &&  
 Objects.equals(name, customer.name) &&  
 Objects.equals(lastname, customer.lastname) &&  
 Objects.equals(company, customer.company) &&  
 Objects.equals(nip, customer.nip) &&  
 Objects.equals(address, customer.address) &&  
 Objects.equals(deliveryAddress, customer.deliveryAddress);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.hash(getId(), name, lastname, company, nip, address, deliveryAddress);  
 }  
}

1. Dimensions.java

package model;  
  
import java.io.Serializable;  
  
public class Dimensions implements Serializable {  
 private double length;  
 private double width;  
 private double height;  
  
 public Dimensions(double length, double width, double height) {  
 this.length = length;  
 this.width = width;  
 this.height = height;  
 }  
  
 public double getLength() {  
 return length;  
 }  
  
 public double getWidth() {  
 return width;  
 }  
  
 public double getHeight() {  
 return height;  
 }  
}

1. ItemList.java

package model;  
  
import java.math.BigDecimal;  
  
public class ItemsList extends AbstractModel {  
 private int quantity;  
 private String name;  
 private int discount; // optional  
 private BigDecimal netTotal;  
 private BigDecimal grossTotal;  
  
 public ItemsList(Integer id, int quantity, String name, int discount, BigDecimal netTotal, BigDecimal grossTotal) {  
 super(id);  
 this.quantity = quantity;  
 this.name = name;  
 this.discount = discount;  
 this.netTotal = netTotal;  
 this.grossTotal = grossTotal;  
 }  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getDiscount() {  
 return discount;  
 }  
  
 public BigDecimal getNetTotal() {  
 return netTotal;  
 }  
  
 public BigDecimal getGrossTotal() {  
 return grossTotal;  
 }  
}

1. Order.java

package model;  
import java.math.BigDecimal;  
import java.math.RoundingMode;  
import java.util.Date;  
import java.util.List;  
public class Order extends AbstractModel {  
 private Date date;  
 private List<ItemsList> itemsList;  
 private Customer client;  
 private Address deliveryAddress;  
  
 public Order(Integer id, Date date, List<model.ItemsList> itemsList, Customer client, Address deliveryAddress) {  
 super(id);  
 this.date = date;  
 this.itemsList = itemsList;  
 this.client = client;  
 this.deliveryAddress = deliveryAddress;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public List<model.ItemsList> getItemsList() {  
 return itemsList;  
 }  
  
 public Customer getClient() {  
 return client;  
 }  
  
 public Address getDeliveryAddress() {  
 return deliveryAddress;  
 }  
  
 public BigDecimal getOrderTotalPrice() {  
 BigDecimal total = BigDecimal.ZERO;  
  
 for (ItemsList item : itemsList) {  
 BigDecimal itemTotal = item.getGrossTotal();  
  
 if (item.getDiscount() > 0) {  
 BigDecimal discount = BigDecimal.valueOf(item.getDiscount()).divide(new BigDecimal(100), 2, RoundingMode.HALF\_UP);  
 itemTotal = itemTotal.subtract(itemTotal.multiply(discount));  
 }  
  
 total = total.add(itemTotal);  
 }  
  
 return total.setScale(2, RoundingMode.HALF\_UP);  
 }  
}

1. Product.java

package model;  
  
import java.math.BigDecimal;  
  
public class Product extends AbstractModel {  
 private String name;  
 private String description; // optional  
 private String sku;  
 private BigDecimal netPrice;  
 private BigDecimal grossPrice;  
 private int tax;  
 private Dimensions dimensions; // optional  
 private Double weight; // optional  
  
 public Product(Integer id, String name, String description, String sku, BigDecimal netPrice, BigDecimal grossPrice, int tax, Dimensions dimensions, Double weight) {  
 super(id);  
 this.name = name;  
 this.description = description;  
 this.sku = sku;  
 this.netPrice = netPrice;  
 this.grossPrice = grossPrice;  
 this.tax = tax;  
 this.dimensions = dimensions;  
 this.weight = weight;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public String getSku() {  
 return sku;  
 }  
  
 public BigDecimal getNetPrice() {  
 return netPrice;  
 }  
  
 public BigDecimal getGrossPrice() {  
 return grossPrice;  
 }  
  
 public int getTax() {  
 return tax;  
 }  
  
 public Dimensions getDimensions() {  
 return dimensions;  
 }  
  
 public Double getWeight() {  
 return weight;  
 }  
  
  
}

1. DateTimeUtil.java

package util;  
  
import javax.swing.\*;  
import javax.swing.text.MaskFormatter;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class DateTimeUtil {  
  
 private static String *DEFAULT\_FORMAT* = "dd-MM-yyyy";  
  
 public static String showDate(Date date) {  
 SimpleDateFormat formatter = new SimpleDateFormat(*DEFAULT\_FORMAT*);  
 return formatter.format(date);  
 }  
  
 public static Date parseDate(String dateString) {  
 if (dateString == null || dateString.trim().isEmpty()) {  
 return null;  
 }  
  
 SimpleDateFormat dateFormat = new SimpleDateFormat(*DEFAULT\_FORMAT*);  
 dateFormat.setLenient(false);  
  
 try {  
 return dateFormat.parse(dateString);  
 } catch (ParseException e) {  
 return null;  
 }  
 }  
  
  
 public static JFormattedTextField createTextFieldWithDataFormat() {  
 MaskFormatter dateMask = null;  
 try {  
 dateMask = new MaskFormatter("##-##-####");  
 dateMask.setPlaceholderCharacter('\_');  
 } catch (ParseException e) {  
 e.printStackTrace();  
 }  
  
 JFormattedTextField dateField = new JFormattedTextField(dateMask);  
 dateField.setColumns(10);  
 return dateField;  
 }  
}

1. FileUtil.java

package util;  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class FileUtil {  
 public static <T> void saveToFile(String fileName, List<T> list) {  
 try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(fileName))) {  
 out.writeObject(list);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 public static <T> List<T> loadFromFile(String fileName) {  
 File file = new File(fileName);  
 if (file.exists()) {  
 try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(file))) {  
 return (List<T>) in.readObject();  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 return new ArrayList<>();  
 }  
  
  
}

1. ValidatorUtil.java

package util;  
  
import javax.swing.\*;  
import java.math.BigDecimal;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.regex.Pattern;  
  
public class ValidatorUtil {  
 public static boolean validateNIP(String nip) {  
 if (nip == null || nip.isEmpty()) {  
 return true;  
 }  
 String nipPattern = "\\d{3}-\\d{2}-\\d{2}-\\d{2}|\\d{10}";  
 return Pattern.matches(nipPattern, nip);  
 }  
  
 public static boolean validatePostalCode(String postalCode) {  
 String postalCodePattern = "\\d{2}-\\d{3}";  
 return Pattern.matches(postalCodePattern, postalCode);  
 }  
  
 public static boolean validateTextField(String text) {  
 return text != null && !text.trim().isEmpty();  
 }  
  
  
  
 public static boolean validateIntRange(int number, int min, int max) {  
 return number >= min && number <= max;  
 }  
  
 public static boolean validatePrice(BigDecimal price) {  
 if (price.compareTo(BigDecimal.ZERO) <= 0) {  
 return false;  
 }  
 if (price.scale() > 2) {  
 return false;  
 }  
 return true;  
 }  
  
 public static int validateWeight(String weightStr) {  
 if (weightStr == null || weightStr.trim().isEmpty()) {  
 return 0;  
 }  
  
 try {  
 double weight = Double.parseDouble(weightStr.trim());  
  
 if (weight <= 0) {  
 return 1;  
 }  
  
 return 0;  
 } catch (NumberFormatException e) {  
 return 2;  
 }  
 }  
  
 public static int validateDimensions(String width, String height, String depth) {  
 String[] dimensions = {width, height, depth};  
 long countProvided = java.util.Arrays.stream(dimensions).filter(d -> d != null && !d.trim().isEmpty()).count();  
 if (countProvided > 0 && countProvided < dimensions.length) {  
 return 1;  
 }  
  
 try {  
 for (String dim : dimensions) {  
 if (dim != null && !dim.trim().isEmpty()) {  
 double dimDouble = Double.parseDouble(dim.trim());  
 if (dimDouble < 0) {  
 return 0;  
 }  
 }  
 }  
 return 3;  
 } catch (NumberFormatException e) {  
 return 2;  
 }  
 }  
}

1. AbstractView.java

package view;  
  
import javax.swing.\*;  
import javax.swing.border.EmptyBorder;  
import javax.swing.table.DefaultTableModel;  
import javax.swing.table.TableModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseAdapter;  
import java.math.BigDecimal;  
  
public abstract class AbstractView extends JPanel {  
 protected JButton addButton = new JButton("Dodaj");;  
 protected JButton removeButton = new JButton("Usuń");;  
 protected JButton filterButton = new JButton("Filtruj");  
 protected JButton resetButton = new JButton("Reset");  
 protected JTable table;  
 public DefaultTableModel tableModel;  
 protected TableRowSorter<TableModel> sorter;  
  
 public AbstractView(String[] columnNames) {  
 tableModel = new DefaultTableModel(columnNames, 0) {  
 @Override  
 public Class<?> getColumnClass(int column) {  
 if (column == findColumn("Suma Zamówień")) {  
 return BigDecimal.class;  
 }  
 return super.getColumnClass(column);  
 }  
 };  
 table = new JTable(tableModel);  
 sorter = new TableRowSorter<>(table.getModel());  
 table.setRowSorter(sorter);  
 JScrollPane listScrollPane = new JScrollPane(table);  
  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new FlowLayout(FlowLayout.*CENTER*));  
 buttonPanel.add(addButton);  
 buttonPanel.add(removeButton);  
  
 setLayout(new BorderLayout());  
 setBorder(new EmptyBorder(10, 10, 10, 10));  
 add(listScrollPane, BorderLayout.*CENTER*);  
 add(buttonPanel, BorderLayout.*SOUTH*);  
  
 table.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*);  
 table.setDefaultEditor(Object.class, null);  
 }  
  
 public void addButtonAction(ActionListener action) {  
 addButton.addActionListener(action);  
 }  
  
 public void removeButtonAction(ActionListener action) {  
 removeButton.addActionListener(action);  
 }  
  
 public void doubleClickAction(MouseAdapter action) {  
 table.addMouseListener(action);  
 }  
  
 public void addActionToFilterButton(ActionListener action) {  
 filterButton.addActionListener(action);  
 }  
  
 public void addActionToResetButton(ActionListener action) {  
 resetButton.addActionListener(action);  
 }  
  
 public abstract void addToView(Object o);  
  
 public void removeFromView(int id) {  
 for (int i = 0; i < tableModel.getRowCount(); i++) {  
 if (tableModel.getValueAt(i, 0).equals(id)) {  
 tableModel.removeRow(i);  
 break;  
 }  
 }  
 }  
  
 public Integer getSelectedElement() {  
 int selectedRow = table.getSelectedRow();  
 if (selectedRow >= 0) {  
 return (Integer) tableModel.getValueAt(selectedRow, 0);  
 }  
 return null;  
 }  
  
 public TableRowSorter<TableModel> getSorter() {  
 return sorter;  
 }  
  
 public JTable getTable() {  
 return table;  
 }  
}

1. AbstractFormView.java

package view;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
  
public abstract class AbstractFormView extends JDialog {  
 private final JButton submitBtn = new JButton("Wyślij");  
 private final JButton cancelBtn = new JButton("Anuluj");  
  
 public AbstractFormView(Frame parent, String title) {  
 super(parent, title, true);  
 getContentPane().setLayout(new BorderLayout());  
 addBtnsToForm();  
 }  
  
 public void addBtnsToForm() {  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new FlowLayout(FlowLayout.*CENTER*));  
 buttonPanel.add(submitBtn);  
 buttonPanel.add(cancelBtn);  
 getContentPane().add(buttonPanel, BorderLayout.*SOUTH*);  
 }  
  
  
 protected void addField(JPanel panel, String label, Component field, GridBagConstraints gbc) {  
 panel.add(new JLabel(label), gbc);  
 gbc.gridx++;  
 if (field instanceof JTextField) {  
 ((JTextField) field).setPreferredSize(new Dimension(250, 20));  
 }  
 panel.add(field, gbc);  
 gbc.gridx = 0;  
 gbc.gridy++;  
 }  
  
 protected abstract void addFieldsToForm();  
 public abstract void clearFormFields();  
  
 public void submitForm(ActionListener actionListener) {  
 submitBtn.addActionListener(actionListener);  
 }  
  
 public void cancelForm(ActionListener actionListener) {  
 cancelBtn.addActionListener(actionListener);  
 }  
}

1. ProductView.java

package view.product;  
  
import model.Product;  
import view.AbstractView;  
  
import javax.swing.\*;  
import javax.swing.event.DocumentEvent;  
import javax.swing.event.DocumentListener;  
import javax.swing.table.TableModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.awt.event.MouseAdapter;  
  
public class ProductView extends AbstractView {  
 JTextField filterField = new JTextField(15);  
  
 public ProductView() {  
 super(new String[]{"ID", "Nazwa", "SKU", "Cena Netto", "Cena Brutto"});  
 JPanel filterPanel = new JPanel();  
 filterPanel.add(new JLabel("Szukaj po nazwie:"));  
 filterPanel.add(filterField);  
 add(filterPanel, BorderLayout.*NORTH*);  
 }  
  
 public JTextField getFilterField() {  
 return filterField;  
 }  
  
  
 @Override  
 public void addToView(Object o) {  
 Product product = (Product) o;  
 tableModel.addRow(new Object[]{product.getId(), product.getName(), product.getSku(), product.getNetPrice(), product.getGrossPrice()});  
 }  
  
 public void searchAction(DocumentListener listener) {  
 filterField.getDocument().addDocumentListener(listener);  
 }  
}

1. ProductFormView.java

package view.product;  
  
import view.AbstractFormView;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class ProductFormView extends AbstractFormView {  
  
 private JTextField nameField = new JTextField();  
 private JTextArea descriptionArea = new JTextArea();  
 private JTextField skuField = new JTextField();  
 private JTextField netPriceField = new JTextField();  
 private JTextField taxField = new JTextField();  
 private JTextField lengthField = new JTextField();  
 private JTextField widthField = new JTextField();  
 private JTextField heightField = new JTextField();  
 private JTextField weightField = new JTextField();  
  
 public ProductFormView(Frame parent) {  
 super(parent, "Wpisz dane nowego produktu");  
 addFieldsToForm();  
 pack();  
 setLocationRelativeTo(parent);  
 }  
  
 public JTextField getNameField() {  
 return nameField;  
 }  
  
 public JTextArea getDescriptionArea() {  
 return descriptionArea;  
 }  
  
 public JTextField getSkuField() {  
 return skuField;  
 }  
  
 public JTextField getNetPriceField() {  
 return netPriceField;  
 }  
  
 public JTextField getTaxField() {  
 return taxField;  
 }  
  
 public JTextField getLengthField() {  
 return lengthField;  
 }  
  
 public JTextField getWidthField() {  
 return widthField;  
 }  
  
 public JTextField getHeightField() {  
 return heightField;  
 }  
  
 public JTextField getWeightField() {  
 return weightField;  
 }  
  
 @Override  
 protected void addFieldsToForm() {  
 JPanel formPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 descriptionArea.setLineWrap(true);  
 descriptionArea.setWrapStyleWord(true);  
 JScrollPane descriptionScrollPane = new JScrollPane(descriptionArea,  
 JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*,  
 JScrollPane.*HORIZONTAL\_SCROLLBAR\_NEVER*);  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
 gbc.anchor = GridBagConstraints.*WEST*;  
 gbc.insets = new Insets(2, 2, 2, 2);  
 gbc.weightx = 1.0;  
  
  
  
 addField(formPanel, "Nazwa:", nameField, gbc);  
 descriptionScrollPane.setPreferredSize(new Dimension(200, 100)); // szerokość x wysokość  
  
 addField(formPanel, "Opis (opcjonalnie):", descriptionScrollPane, gbc); // Specjalne traktowanie dla JTextArea z JScrollPane  
  
 addField(formPanel, "SKU:", skuField, gbc);  
  
 addField(formPanel, "Cena NETTO:", netPriceField, gbc);  
  
 addField(formPanel, "Podatek %:", taxField, gbc);  
  
 addField(formPanel, "Długość cm (opcjonalnie):", lengthField, gbc);  
  
 addField(formPanel, "Szerokość cm (opcjonalnie):", widthField, gbc);  
  
 addField(formPanel, "Wysokość cm (opcjonalnie):", heightField, gbc);  
  
 addField(formPanel, "Waga kg (opcjonalnie):", weightField, gbc);  
 getContentPane().add(formPanel, BorderLayout.*CENTER*);  
 }  
  
 @Override  
 public void clearFormFields() {  
 nameField.setText("");  
 descriptionArea.setText("");  
 skuField.setText("");  
 netPriceField.setText("");  
 taxField.setText("");  
 lengthField.setText("");  
 widthField .setText("");  
 heightField.setText("");  
 weightField.setText("");  
 }  
}

1. OrderView.java

package view.order;  
  
import model.Customer;  
import model.Order;  
import util.DateTimeUtil;  
import view.AbstractView;  
  
import javax.swing.\*;  
import javax.swing.event.PopupMenuListener;  
import java.awt.\*;  
import java.text.SimpleDateFormat;  
  
public class OrderView extends AbstractView {  
  
 private SimpleDateFormat dataFormatter = new SimpleDateFormat("dd-MM-yyyy");  
 private JFormattedTextField startDateField = DateTimeUtil.createTextFieldWithDataFormat();  
 private JFormattedTextField endDateField = DateTimeUtil.createTextFieldWithDataFormat();;  
 private JComboBox<Customer> customerComboBox = new JComboBox<>();  
 private JTextField minOrderValueField = new JTextField();  
 private JTextField maxOrderValueField = new JTextField();  
  
  
 public SimpleDateFormat getDataFormatter() {  
 return dataFormatter;  
 }  
  
 public JFormattedTextField getStartDateField() {  
 return startDateField;  
 }  
  
 public JFormattedTextField getEndDateField() {  
 return endDateField;  
 }  
  
 public JComboBox<Customer> getCustomerComboBox() {  
 return customerComboBox;  
 }  
  
 public JTextField getMinOrderValueField() {  
 return minOrderValueField;  
 }  
  
 public JTextField getMaxOrderValueField() {  
 return maxOrderValueField;  
 }  
  
 public OrderView() {  
 super(new String[]{"ID", "Data złożenia", "Cena", "Zamawiający"});  
  
 customerComboBox.setPreferredSize(new Dimension(150, 30));  
 minOrderValueField.setPreferredSize(new Dimension(100, minOrderValueField.getPreferredSize().height));  
 maxOrderValueField.setPreferredSize(new Dimension(100, maxOrderValueField.getPreferredSize().height));  
  
 addFiltersToView();  
  
 dataFormatter.setLenient(false);  
 }  
  
 private void addFiltersToView() {  
 JPanel filterPanel = new JPanel();  
 filterPanel.setLayout(new BoxLayout(filterPanel, BoxLayout.Y\_AXIS));  
  
 JPanel clientPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 clientPanel.add(new JLabel("Klient:"));  
 clientPanel.add(customerComboBox);  
  
  
 JPanel startDatePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 startDatePanel.add(new JLabel("Data rozpoczęcia:"));  
 startDatePanel.add(startDateField);  
  
 JPanel endDatePanel = new JPanel(new FlowLayout(FlowLayout.*LEFT*));  
 endDatePanel.add(new JLabel("Data zakończenia:"));  
 endDatePanel.add(endDateField);  
  
 JPanel minOrderValuePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 minOrderValuePanel.add(new JLabel("Min. wartość zamówienia:"));  
 minOrderValuePanel.add(minOrderValueField);  
  
 JPanel maxOrderValuePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 maxOrderValuePanel.add(new JLabel("Max. wartość zamówienia:"));  
 maxOrderValuePanel.add(maxOrderValueField);  
  
 JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 buttonsPanel.add(filterButton);  
 buttonsPanel.add(resetButton);  
  
 filterPanel.add(clientPanel);  
 filterPanel.add(startDatePanel);  
 filterPanel.add(endDatePanel);  
 filterPanel.add(maxOrderValuePanel);  
 filterPanel.add(minOrderValuePanel);  
 filterPanel.add(buttonsPanel);  
  
 JPanel centeredFilterPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.weightx = 1;  
 gbc.weighty = 1;  
 gbc.anchor = GridBagConstraints.CENTER;  
 centeredFilterPanel.add(filterPanel, gbc);  
  
 add(centeredFilterPanel, BorderLayout.NORTH);  
 }  
  
 @Override  
 public void addToView(Object o) {  
 Order order = (Order) o;  
 tableModel.addRow(new Object[]{order.getId(), DateTimeUtil.showDate(order.getDate()), order.getOrderTotalPrice(), order.getClient()});  
 }  
  
 public void addActionToCustomerComboBox(PopupMenuListener actionListener) {  
 customerComboBox.addPopupMenuListener(actionListener);  
 }  
}

1. OrderFormView.java

package view.order;  
  
import model.Customer;  
import model.Product;  
import util.DateTimeUtil;  
import view.AbstractFormView;  
  
import javax.swing.\*;  
import javax.swing.event.PopupMenuListener;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.ActionListener;  
import java.awt.\*;  
import javax.swing.event.TableModelListener;  
  
public class OrderFormView extends AbstractFormView {  
  
 private JFormattedTextField orderDateField = DateTimeUtil.createTextFieldWithDataFormat();  
  
 private JComboBox<Customer> customerComboBox = new JComboBox<>();  
 private JTable productTable;  
 private DefaultTableModel productTableModel;  
 public JButton selectButton = new JButton("Dodaj");;  
 public JComboBox<Product> productComboBox = new JComboBox<>();  
 public JButton addButton = new JButton("Dodaj Produkt");  
 public JButton removeButton = new JButton("Usuń");  
 private JTextField deliveryStreetField = new JTextField();  
 private JTextField deliveryHouseNumberField = new JTextField();  
 private JTextField deliveryApartmentNumberField = new JTextField();  
 private JTextField deliveryCityField = new JTextField();  
 private JTextField deliveryPostalCodeField = new JTextField();  
 private JTextField deliveryStateField = new JTextField();  
 private JTextField deliveryCountryField = new JTextField();  
  
 public OrderFormView(Frame parent) {  
 super(parent, "Dodaj nowe zamówienie");  
 addFieldsToForm();  
 pack();  
 setLocationRelativeTo(parent);  
 }  
  
 public JComboBox<Customer> getCustomerComboBox() {  
 return customerComboBox;  
 }  
  
 public JComboBox<Product> getProductComboBox() {  
 return productComboBox;  
 }  
  
 public JTable getProductTable() {  
 return productTable;  
 }  
  
 public DefaultTableModel getproductTableModel() {  
 return productTableModel;  
 }  
  
 public JTextField getDeliveryStreetField() {  
 return deliveryStreetField;  
 }  
  
  
 public JTextField getDeliveryHouseNumberField() {  
 return deliveryHouseNumberField;  
 }  
  
 public JTextField getDeliveryApartmentNumberField() {  
 return deliveryApartmentNumberField;  
 }  
  
 public JTextField getDeliveryCityField() {  
 return deliveryCityField;  
 }  
  
 public JTextField getDeliveryPostalCodeField() {  
 return deliveryPostalCodeField;  
 }  
  
 public JTextField getDeliveryStateField() {  
 return deliveryStateField;  
 }  
  
 public JTextField getDeliveryCountryField() {  
 return deliveryCountryField;  
 }  
  
 public JTextField getOrderDateField() {  
 return orderDateField;  
 }  
  
 @Override  
 protected void addFieldsToForm() {  
 JPanel formPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.fill = GridBagConstraints.HORIZONTAL;  
 gbc.anchor = GridBagConstraints.WEST;  
 gbc.insets = new Insets(2, 2, 2, 2);  
 gbc.weightx = 1.0;  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
  
 addField(formPanel, "Data złożenia zamówienia::", orderDateField, gbc);  
 customerComboBox.setPreferredSize(new Dimension(250, 20));  
 addField(formPanel, "Klient:", customerComboBox, gbc);  
 addField(formPanel, "Adres dostawy - Ulica:", deliveryStreetField, gbc);  
 addField(formPanel, "Adres dostawy - Numer domu:", deliveryHouseNumberField, gbc);  
 addField(formPanel, "Adres dostawy - Numer mieszkania:", deliveryApartmentNumberField, gbc);  
 addField(formPanel, "Adres dostawy - Miejscowość:", deliveryCityField, gbc);  
 addField(formPanel, "Adres dostawy - Kod pocztowy:", deliveryPostalCodeField, gbc);  
 addField(formPanel, "Adres dostawy - Województwo:", deliveryStateField, gbc);  
 addField(formPanel, "Adres dostawy - Państwo:", deliveryCountryField, gbc);  
  
 getContentPane().add(formPanel, BorderLayout.CENTER);  
 addProductTableToView();  
 productComboBox.setPreferredSize(new Dimension(350, 20));  
 }  
  
 private void addProductTableToView() {  
 JPanel productPanel = new JPanel(new BorderLayout());  
 String[] columnNames = {"ID", "Nazwa", "Netto", "Brutto", "Ilość", "Rabat", "Suma"};  
 productTableModel = new DefaultTableModel(null, columnNames) {  
 public boolean isCellEditable(int row, int column) {  
 return column == 4 || column == 5;  
 }  
 };  
 productTable = new JTable(productTableModel);  
 JScrollPane scrollPane = new JScrollPane(productTable);  
 productPanel.add(scrollPane, BorderLayout.CENTER);  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));  
 buttonPanel.add(addButton);  
 buttonPanel.add(removeButton);  
 productPanel.add(buttonPanel, BorderLayout.SOUTH);  
 getContentPane().add(productPanel, BorderLayout.EAST);  
  
 addActionToAddButton(e -> createDialogWithProductComboBox());  
 }  
  
 @Override  
 public void clearFormFields() {  
 deliveryStreetField.setText("");  
 deliveryHouseNumberField.setText("");  
 deliveryApartmentNumberField.setText("");  
 deliveryCityField.setText("");  
 deliveryPostalCodeField.setText("");  
 deliveryStateField.setText("");  
 deliveryCountryField.setText("");  
 orderDateField.setText("");  
 productTableModel.setRowCount(0);  
 }  
  
 private void createDialogWithProductComboBox() {  
 JDialog dialog = new JDialog(this, "Dodaj produkt do zamówienia", true);  
 dialog.setLayout(new FlowLayout());  
 dialog.add(productComboBox);  
  
 dialog.add(selectButton);  
 dialog.pack();  
 dialog.setLocationRelativeTo(this);  
 dialog.setVisible(true);  
 }  
  
 public void addActionToAddButton(ActionListener actionListener) {  
 addButton.addActionListener(actionListener);  
 }  
  
 public void addActionToSelectButton(ActionListener actionListener) {  
 selectButton.addActionListener(actionListener);  
 }  
  
 public void addActionToRemoveButton(ActionListener actionListener) {  
 removeButton.addActionListener(actionListener);  
 }  
  
 public void addActionToProductTableModel(TableModelListener actionListener) {  
 productTableModel.addTableModelListener(actionListener);  
 }  
  
 public void addActionToCustomerComboBox(PopupMenuListener actionListener) {  
 customerComboBox.addPopupMenuListener(actionListener);  
 }  
  
}

1. CustomerView.java

package view.customer;  
  
import model.Customer;  
import util.DateTimeUtil;  
import view.AbstractView;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.math.BigDecimal;  
import java.text.SimpleDateFormat;  
import java.util.Comparator;  
  
  
public class CustomerView extends AbstractView {  
  
 private JFormattedTextField startDateField = DateTimeUtil.createTextFieldWithDataFormat();  
 private JFormattedTextField endDateField = DateTimeUtil.createTextFieldWithDataFormat();;  
 private JTextField minOrderValueField = new JTextField();  
  
 public JFormattedTextField getStartDateField() {  
 return startDateField;  
 }  
  
 public JFormattedTextField getEndDateField() {  
 return endDateField;  
 }  
  
 public JTextField getMinOrderValueField() {  
 return minOrderValueField;  
 }  
  
  
 public CustomerView() {  
 super(new String[]{"ID", "Imię", "Nazwisko", "Nazwa firmy", "NIP"});  
 addFiltersToView();  
 setStylesToFilters();  
 }  
  
 @Override  
 public void addToView(Object o) {  
 Customer customer = (Customer) o;  
 tableModel.addRow(new Object[]{customer.getId(), customer.getName(), customer.getLastname(), customer.getCompany(), customer.getNip()});  
 }  
  
 private void addFiltersToView() {  
 JPanel filterPanel = new JPanel();  
 filterPanel.setLayout(new BoxLayout(filterPanel, BoxLayout.Y\_AXIS));  
  
 JPanel startDatePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 startDatePanel.add(new JLabel("Data rozpoczęcia:"));  
 startDatePanel.add(startDateField);  
  
 JPanel endDatePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 endDatePanel.add(new JLabel("Data zakończenia:"));  
 endDatePanel.add(endDateField);  
  
 JPanel minOrderValuePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 minOrderValuePanel.add(new JLabel("Min. wartość zamówienia:"));  
 minOrderValuePanel.add(minOrderValueField);  
  
 JPanel buttonsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));  
 buttonsPanel.add(filterButton);  
 buttonsPanel.add(resetButton);  
  
 filterPanel.add(startDatePanel);  
 filterPanel.add(endDatePanel);  
 filterPanel.add(minOrderValuePanel);  
 filterPanel.add(buttonsPanel);  
  
 JPanel centeredFilterPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.weightx = 1;  
 gbc.weighty = 1;  
 gbc.anchor = GridBagConstraints.CENTER;  
 centeredFilterPanel.add(filterPanel, gbc);  
  
 add(centeredFilterPanel, BorderLayout.NORTH);  
 }  
  
 private void setStylesToFilters() {  
 minOrderValueField.setPreferredSize(new Dimension(100, minOrderValueField.getPreferredSize().height));  
 }  
  
  
  
}

1. CustomerFormView.java

package view.customer;  
  
import view.AbstractFormView;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
  
public class CustomerFormView extends AbstractFormView {  
 private JTextField firstNameField = new JTextField();  
 private JTextField lastNameField = new JTextField();  
 private JTextField companyNameField = new JTextField();  
 private JTextField nipField = new JTextField();  
 private JTextField streetField = new JTextField();  
 private JTextField buildingNumberField = new JTextField();  
 private JTextField apartmentNumberField = new JTextField();  
 private JTextField cityField = new JTextField();  
 private JTextField postalCodeField = new JTextField();  
 private JTextField stateField = new JTextField();  
 private JTextField countryField = new JTextField();  
  
 private JRadioButton deliveryAddressYes = new JRadioButton("Tak");  
 private JRadioButton deliveryAddressNo = new JRadioButton("Nie", true);  
 private ButtonGroup deliveryAddressGroup = new ButtonGroup();  
  
 // delivery address  
 private JTextField deliveryStreetField = new JTextField();  
 private JTextField deliveryHouseNumberField = new JTextField();  
 private JTextField deliveryApartmentNumberField = new JTextField();  
 private JTextField deliveryCityField = new JTextField();  
 private JTextField deliveryPostalCodeField = new JTextField();  
 private JTextField deliveryStateField = new JTextField();  
 private JTextField deliveryCountryField = new JTextField();  
  
 private JPanel deliveryAddressPanel = new JPanel(new GridBagLayout());  
  
 public CustomerFormView(Frame parent) {  
 super(parent, "Wprowadź dane klienta");  
 addFieldsToForm();  
 pack();  
 setLocationRelativeTo(parent);  
 setMinimumSize(getSize());  
 }  
  
 public JTextField getNameField() {  
 return firstNameField;  
 }  
  
 public JTextField getLastNameField() {  
 return lastNameField;  
 }  
  
 public JTextField getCompanyField() {  
 return companyNameField;  
 }  
  
 public JTextField getNipField() {  
 return nipField;  
 }  
  
 public JTextField getStreetField() {  
 return streetField;  
 }  
  
 public JTextField getHouseNumberField() {  
 return buildingNumberField;  
 }  
  
 public JTextField getApartmentNumberField() {  
 return apartmentNumberField;  
 }  
  
 public JTextField getCityField() {  
 return cityField;  
 }  
  
 public JTextField getPostalCodeField() {  
 return postalCodeField;  
 }  
  
 public JTextField getStateField() {  
 return stateField;  
 }  
  
 public JTextField getCountryField() {  
 return countryField;  
 }  
  
 public JTextField getDeliveryStreetField() {  
 return deliveryStreetField;  
 }  
  
 public JTextField getDeliveryHouseNumberField() {  
 return deliveryHouseNumberField;  
 }  
  
 public JTextField getDeliveryApartmentNumberField() {  
 return deliveryApartmentNumberField;  
 }  
  
 public JTextField getDeliveryCityField() {  
 return deliveryCityField;  
 }  
  
 public JTextField getDeliveryPostalCodeField() {  
 return deliveryPostalCodeField;  
 }  
  
 public JTextField getDeliveryStateField() {  
 return deliveryStateField;  
 }  
  
 public JTextField getDeliveryCountryField() {  
 return deliveryCountryField;  
 }  
  
 public JRadioButton getdeliveryAddressYes() {  
 return deliveryAddressYes;  
 }  
  
 public JPanel getDeliveryAddressPanel() {  
 return deliveryAddressPanel;  
 }  
  
 public void addFieldsToForm() {  
 JPanel formPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 gbc.fill = GridBagConstraints.HORIZONTAL;  
 gbc.anchor = GridBagConstraints.WEST;  
 gbc.insets = new Insets(2, 2, 2, 2);  
 gbc.weightx = 1.0;  
  
 addField(formPanel, "Imię:", firstNameField, gbc);  
 addField(formPanel, "Nazwisko:", lastNameField, gbc);  
 addField(formPanel, "Nazwa firmy (opcjonalne):", companyNameField, gbc);  
 addField(formPanel, "NIP (opcjonalne):", nipField, gbc);  
 addField(formPanel, "Ulica:", streetField, gbc);  
 addField(formPanel, "Numer budynku:", buildingNumberField, gbc);  
 addField(formPanel, "Numer mieszkania (opcjonalne):", apartmentNumberField, gbc);  
 addField(formPanel, "Miasto:", cityField, gbc);  
 addField(formPanel, "Kod pocztowy:", postalCodeField, gbc);  
 addField(formPanel, "Województwo:", stateField, gbc);  
 addField(formPanel, "Kraj:", countryField, gbc);  
  
 addDeliveryAddressRadioBtns(formPanel, gbc);  
  
 addField(formPanel, "Adres dostawy - Ulica:", deliveryStreetField, gbc);  
 addField(formPanel, "Adres dostawy - Numer budynku:", deliveryHouseNumberField, gbc);  
 addField(formPanel, "Adres dostawy - Numer mieszkania (opcjonalne):", deliveryApartmentNumberField, gbc);  
 addField(formPanel, "Adres dostawy - Miasto:", deliveryCityField, gbc);  
 addField(formPanel, "Adres dostawy - Kod pocztowy:", deliveryPostalCodeField, gbc);  
 addField(formPanel, "Adres dostawy - Województwo:", deliveryStateField, gbc);  
 addField(formPanel, "Adres dostawy - Kraj:", deliveryCountryField, gbc);  
 setDeliveryAddressFieldsEnabled(false);  
 getContentPane().add(formPanel, BorderLayout.CENTER);  
 }  
  
  
  
 private void addDeliveryAddressRadioBtns(JPanel panel, GridBagConstraints gbc) {  
 gbc.gridwidth = 1;  
 panel.add(new JLabel("Inny adres dostawy:"), gbc);  
  
 deliveryAddressGroup.add(deliveryAddressYes);  
 deliveryAddressGroup.add(deliveryAddressNo);  
 JPanel deliveryOptions = new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));  
 deliveryOptions.add(deliveryAddressYes);  
 deliveryOptions.add(deliveryAddressNo);  
  
 gbc.gridx = 1;  
 gbc.gridwidth = 2;  
 panel.add(deliveryOptions, gbc);  
  
 gbc.gridwidth = 1;  
 gbc.gridx = 0;  
 gbc.gridy++;  
  
 }  
  
 public void clearFormFields() {  
 firstNameField.setText("");  
 lastNameField.setText("");  
 companyNameField.setText("");  
 nipField.setText("");  
 streetField.setText("");  
 buildingNumberField.setText("");  
 apartmentNumberField.setText("");  
 cityField.setText("");  
 postalCodeField.setText("");  
 stateField.setText("");  
 countryField.setText("");  
 deliveryStreetField.setText("");  
 deliveryHouseNumberField.setText("");  
 deliveryApartmentNumberField.setText("");  
 deliveryCityField.setText("");  
 deliveryPostalCodeField.setText("");  
 deliveryStateField.setText("");  
 deliveryCountryField.setText("");  
 }  
  
 public void setDeliveryAddressFieldsEnabled(boolean enabled) {  
 deliveryStreetField.setEnabled(enabled);  
 deliveryHouseNumberField.setEnabled(enabled);  
 deliveryApartmentNumberField.setEnabled(enabled);  
 deliveryCityField.setEnabled(enabled);  
 deliveryPostalCodeField.setEnabled(enabled);  
 deliveryStateField.setEnabled(enabled);  
 deliveryCountryField.setEnabled(enabled);  
 }  
  
 public void showDeliveryPanelYes(ActionListener actionListener) {  
 deliveryAddressYes.addActionListener(actionListener);  
 }  
  
 public void showDeliveryPanelNo(ActionListener actionListener) {  
 deliveryAddressNo.addActionListener(actionListener);  
 }  
  
}

**Lista okienek w aplikacji**

1. Tabelka z klientami

Obraz zawierający tekst, zrzut ekranu, numer, Równolegle

Opis wygenerowany automatycznie

1. Dodawanie klienta

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

1. Wyświetlanie danych klienta

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

1. Lista produktów.

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, numer

Opis wygenerowany automatycznie

1. Dodawanie produktów

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, numer

Opis wygenerowany automatycznie

1. Szczegóły produktu.

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

1. Lista zamówień

Obraz zawierający tekst, zrzut ekranu, numer, Równolegle

Opis wygenerowany automatycznie

1. Dodawanie zamówień

Obraz zawierający tekst, zrzut ekranu, numer, Równolegle

Opis wygenerowany automatycznie

1. Dodawanie produktu do zamówienia

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

1. Szczegóły zamówienia

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie