Національний університет “Києво-Могилянська академія”

**Звіт до лабораторної роботи №2**

З дисципліни: Вступ до програмування

**Автоматизоване робоче місце**

Виконали : Студенти ФІ,ІПЗ-1

Пащук Назар Михайлович

Буковський Станіслав Валерійович

Викладач:

Кирієнко Оксана Валентинівна

**Програма**

Необхідно автоматизувати роботу невеликого підприємства по роботі з складом.

Існує декілька груп товарів (наприклад: Продовольчі, непродовольчі...). В кожній групі товарів існують конкретні товари (наприклад: борошно, гречка ...). У кожного товару є наступні властивості - назва, опис, виробник, кількість на складі, ціна за одиницю. Група товарів містить наступні властивості - назва, опис.

**Завдання**

Реалізувати:

1. Реалізувати графічний інтерфейс користувача.
2. Збереження даних в файл/файли. Один з варіантів: Існує файл в якому знаходяться назви всіх груп товарів. Товари з кожної групи товарів знаходяться в окремому файлі.
3. Назва товару – унікальна.
4. Назва групи товарів - унікальна.
5. Реалізувати додавання/редагування/видалення групи товарів - при видаленні групи товарів, видаляти і всі товари.
6. Реалізувати додавання/редагування/видалення товару в групу товарів.
7. Реалізувати інтерфейс додавання товару, інтерфейс списання товару
8. Пошук товару.
9. Вивід статистичних даних: вивід всіх товарів з інформацією по складу, вивід усіх товарів по групі товарів з інформацією, загальна вартість товару на складі, загальна вартість товарів в групі товарів.
10. До роботи додати звіт про виконання роботи з описом розподілу ролей.

**Розподіл задач**

Буковський Станіслав Валерійович:

* Реалізація графічного інтерфейсу;
* Діалог з користувачем;

Пащук Назар Михайлович:

* Реалізація логіки програми;
* Тестування коду;

**Опис класів і методів**

1. Group.java: клас груп продуктів, що реалізують відповідний функціонал

* addGroup(String groupName, String groupDescription) - метод для додавання групи
* editGroup(int index, String newGroupName, String newGroupDescription) - метод для редагування групи
* deleteGroup(String groupName) - метод для видалення групи
* remove(String groupName) - методи для видалення товарів при видаленні групи
* sortByName() - метод для сортування груп за назвою
* toString() - об’єкт в стрічку
* set/get Groups() - геттери і сеттери
* set/get GroupName() - геттери і сеттери
* set/get GroupDescription() - геттери і сеттери

2. Product.java:

* addProduct(String groupNameInProduct, String productName, String productDescription, String producer, int productQuantityOnStock, double productPrice) - додати продукт
* editProduct( String groupNameInProduct, String productName, String productDescription, String producer, int productQuantityOnStock, double productPrice) - редагувати продукт
* delete(int index) - видалити продукт за індексом
* toString() - об’єкт в стрічку
* addToStock(String productNameToAdd,int quantity) - для додавання продуктів на склад
* writeOffFromStock(String productNameToWriteOff, int quantity) - для списання продуктів з складу
* totalProductCost() - загальна ціна певного продукта на складі
* valueOfProductsOnStock() - загальна ціна продуктів на складі
* sortByName() - метод для сортування за іменем групи
* get/set for groupNameInProduct, productName,productDescription, producer, productQuantityOnStock, productPrice - геттери і сеттери;

3. DataIO.java

* readInfoFromFile() - читає дані з файлу
* readGroups() - читає групи з файлу
* readProducts() - читає продукти з складу
* writeInfoToFile() - записує інформацію в файл
* writeGroups() - записує групу в файл
* writeProducts() - записує продукти в файл

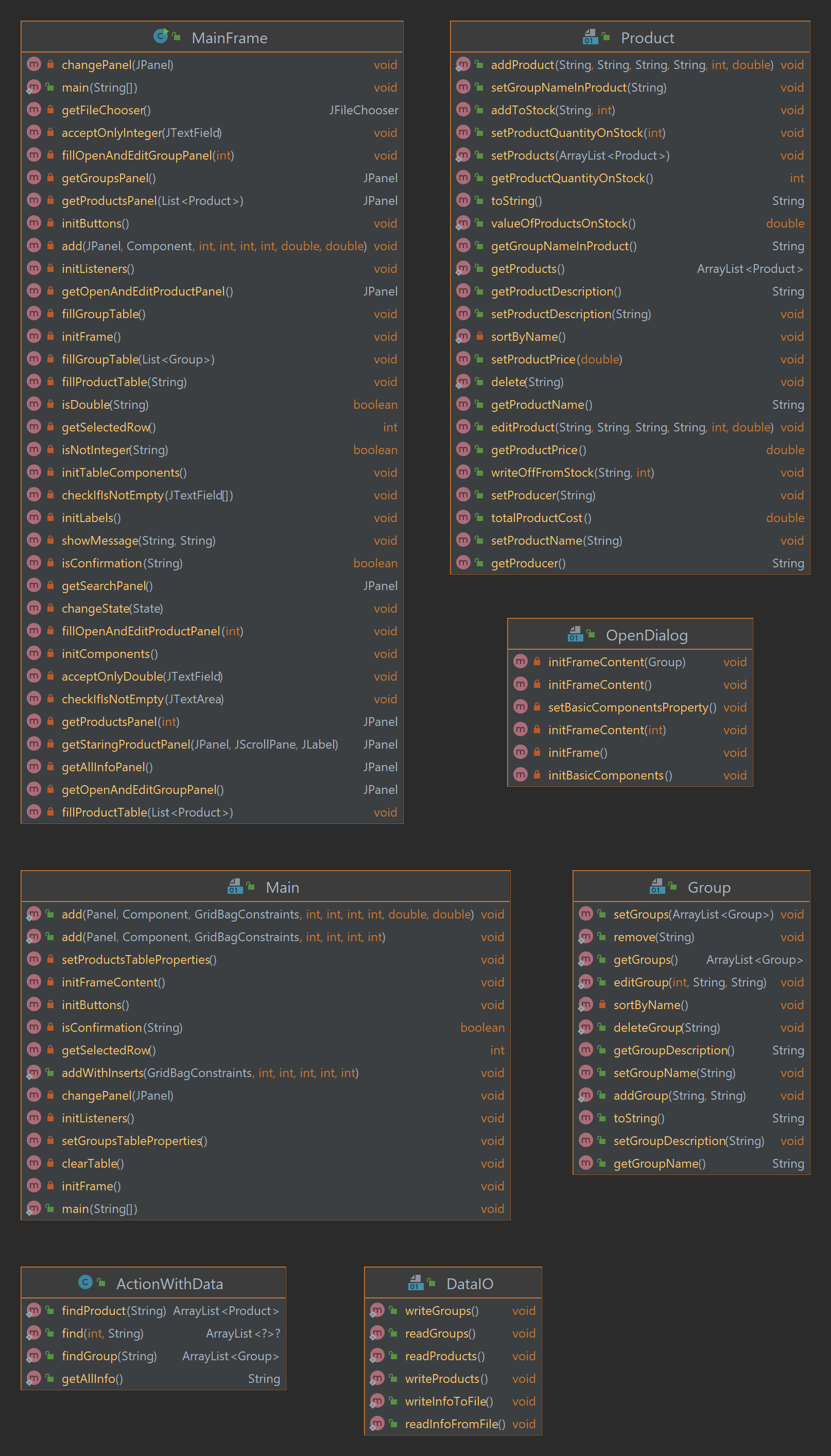
4. ActionWithData.java

* find(int choose, String whatToFind): знаходить файл або групу за стрічкою
* findGroup(String whatToFind):знаходить групу за стрічкою
* findProduct(String whatToFind):знаходить продукт за стрічкою

5. MainFrame.java: клас наслідує JFrame та є головним і єдиним класом для створення GUI.

* MainFrame() – конструктор класу, який ініціалізує сам фрейм та всі елементи такі як кнопки, поля, написи й інші.
* initComponents() – метод, ініціалізує всі кмопненти.
* getOpenAndEditGroupPanel() – метод створює та повертає JPanel для відкриття та редагування групи товарів.
* fillOpenAndEditGroupPanel(int index)
* getSearchPanel() – метод, створює та повертає JPanel для пошук груп творів або товарів.
* getGroupsPanel() – метод створює та повертає JPanel для груп товарів.
* fillGroupTable() – метод наповнює табличку групами товарів, які є в файлі.
* fillGroupTable(List<Group> groups) – метод наповнює табличку групами товарів, які є списку groups.
* fillProductTable(String group) – метод наповнює табличку товарами, які є в групі під назвою group.
* fillProductTable(List<Product> products) – метод наповнює табличку товарами, які є в спику products.
* getOpenAndEditProductPanel() – метод створює та повертає JPanel для вікриття та редагування товару.
* getProductsPanel(List<Product> products) – метод створює та повертає JPanel, яка наповнена товарами з списку products.
* initTableComponents() – метод, ініціалізує елементи які зв’язані з табличками груп товарів та самих товарів.
* initButtons() – метод ініціалізує всі кнопки.
* initListeners() – метод додає AcrionListener для кожної кнопки.
* checkIfIsNotEmpty(JTextField[] textFields) – метод кидає RuntimeException, якщо якесь поле з масиву textField – не містить тексту.
* checkIfIsNotEmpty(JTextArea textArea) – метод кидає RuntimeException, якщо textArea – не містить тексту.
* showMessage(String text, String title) – метод показує інформаційне вікно з текстом text та заголовком title.
* getFileChooser() – метод повертає JFileChooser з фільтром для текстових файлів.
* initLabels() – метод ініціалізує всі текстові написи
* initFrame() – метод ініціалізує JFrame
* add(JPanel panel, Component component, GridBagConstraints gbc, int x, int y, int width, int height, double wx, double wy, int insert) – метод, додає на panel, яка керується GridBagLayout, за координатами x, y та з вагою wx і wy.
* changePanel(JPanel panel) – метод встановлює на фреймі нову панель panel.
* changeState(State newCurrentState) – метод встановлює новий стан та зберігає попередній.
* getSelectedRow() – метод повертає -1 якщо користувач не обрав рядок в таблиці, в іншому випадку індекс вибраного рядку.
* isConfirmation(String text) – метод повертає true – якщо користувач підтвердив дію на спеціальному вікні, false – якщо ж ні
* isInteger(String answer) – метод повертає true –якщо answer можна претворити на int, false – якщо ж ні.
* isDouble(String answer) ) – метод повертає true –якщо answer можна претворити на double, false – якщо ж ні.
* acceptOnlyInteger(JTextField textField) – встановлює для textField можливість вводу тільки чисел
* acceptOnlyDouble(JTextField textField) – встановлює для textField можливість вводу тільки чисел та крапки.

**Діаграми класів**

****

**Проблеми в роботі**

У процесі виконання даної роботи вникало безліч невагомих проблем наприклад питання як зберігати інформацію про товари чи як саму організувати GUI елементи. Також виникали більш важливіші проблеми такі, як неможливість використання одних екземплярів нащадків класу JComponent таких, як JButton, JLable, JTextField та інших. Вирішенням стало створення методів які повертають JPanel з потрібними елементами. Такі метод викликаються тільки тоді коли треба, а інші панелі стираються.

Одна з проблем, які виникала найчастіше: низька швидкість зв’язку, через що важко було працювати в режимі онлайн та обмінюватись данними.

**Висновки**

Написання цієї програми покращило нашу командну роботу, допомогло розвинути навички в програмуванні, покращити якість коду та пристосування до командної роботи. Також виконання роботи помогли розвинути розуміння хорошого користувацького інтерфейсу та вміння використання бібліотеки Swing. Написання класів для читання та запису інформації в файли допомогло краще вивчити IO-класи.

**Код програми**

ActionWithData.java

*package* source;  
  
*import* java.util.ArrayList;  
*import* java.util.stream.Collectors;  
*import* java.util.stream.Stream;  
  
*public class* ActionWithData {  
 *public static* ArrayList<?> find(*int* choose, String whatToFind){  
 *if*(choose == 1)  
 *return* findGroup(whatToFind);  
 *if*(choose == 2)  
 *return* findProduct(whatToFind);  
 *return null*;  
 }  
 *public static* ArrayList<Group> findGroup(String whatToFind){  
 ArrayList<Group> result = *new* ArrayList<>();  
 *for*(Group group: Group.getGroups()){  
 *if*(group.getGroupName().contains(whatToFind)){  
 result.add(group);  
 *continue*;  
 }  
 *if*(group.getGroupDescription().contains(whatToFind))  
 result.add(group);  
 }  
 *if*(result.isEmpty())  
 *throw new* RuntimeException("Matches not found");  
 *return* result;  
 }  
 *public static* ArrayList<Product> findProduct(String whatToFind){  
 ArrayList<Product> result = *new* ArrayList<>();  
 *for*(Product product: Product.getProducts()) {  
 *if* (product.getGroupNameInProduct().contains(whatToFind)) {  
 result.add(product);  
 *continue*;  
 }  
 *if* (product.getProductName().contains(whatToFind)){  
 result.add(product);  
 *continue*;  
 }  
 *if* (product.getProductDescription().contains(whatToFind)){  
 result.add(product);  
 }  
 }  
 *if*(result.isEmpty())  
 *throw new* RuntimeException("Matches not found");  
 *return* result;  
 }  
  
 *public static* String getAllInfo() {  
 StringBuilder res = *new* StringBuilder();  
 res.append("The amount of products groups: ").append(Group.getGroups().size()).append("\n");  
 res.append("The amount of products: ").append(Product.getProducts().size()).append("\n");  
 res.append("The overall price: ").append(Product.valueOfProductsOnStock()).append("\n\n\n");  
 *int* counter = 1;  
 *for*(Group group: Group.getGroups()) {  
  
 res.append(counter).append(") ").append(group.getGroupName()).append(":").append("\n");  
 res.append(" - Descriptions: ").append(group.getGroupDescription()).append("\n");  
 res.append(" - The amount of products: ").append(Product.getProducts().stream().filter(p -> p.getGroupNameInProduct().equalsIgnoreCase(group.getGroupName())).mapToInt(Product::getProductQuantityOnStock).sum()).append("\n");  
 res.append(" - The overall price: ").append(Product.getProducts().stream().filter(p -> p.getGroupNameInProduct().equalsIgnoreCase(group.getGroupName())).mapToDouble(p -> p.getProductPrice() \* p.getProductQuantityOnStock()).sum()).append("\n");  
 res.append(" - Products:").append("\n");  
 *int* productCounter = 1;  
 *for*(Product product: Product.getProducts().stream().filter(p -> p.getGroupNameInProduct().equalsIgnoreCase(group.getGroupName())).collect(Collectors.toSet())) {  
 res.append(" ").append(productCounter).append(") ").append(product.getProductName()).append(":").append("\n");  
 res.append(" - Name: ").append(product.getProductName()).append("\n");  
 res.append(" - Descriptions: ").append(product.getProductDescription()).append("\n");  
 res.append(" - Producer: ").append(product.getProducer()).append("\n");  
 res.append(" - The amount: ").append(product.getProductQuantityOnStock()).append("\n");  
 res.append(" - The price per one: ").append(product.getProductPrice()).append("\n");  
 res.append(" - The overall price: ").append(product.totalProductCost()).append("\n");  
 productCounter++;  
 res.append("\n");  
 }  
 res.append("\n\n");  
 counter++;  
 }  
 *return* res.toString();  
 }  
}

DataIO.java

*package* source;  
  
*import* java.io.\*;  
*import* java.util.Scanner;  
  
*public class* DataIO {  
 *private final static* File ***groupList*** = *new* File("src/resources/groupList.txt");  
 *private final static* File ***productList*** = *new* File("src/resources/productList.txt");  
  
 */\*\*  
 \* Method that read groups and info from file(use when program begin)  
 \* \*/  
 public static void* readInfoFromFile(){  
 readGroups();  
 readProducts();  
 }  
 *public static void* readGroups(){  
 *try*{  
 Scanner scanner = *new* Scanner(***groupList***);  
 *while*(scanner.hasNext()){  
 String[] groupInfo = scanner.nextLine().split("[#%]");  
 Group.addGroup(groupInfo[1],groupInfo[2]);  
 }  
 } *catch*(IOException e){  
 *throw new* RuntimeException(e);  
 }  
 }  
 *public static void* readProducts(){  
 *try*{  
 Scanner scanner = *new* Scanner(***productList***);  
 *while*(scanner.hasNext()){  
 String[] productInfo = scanner.nextLine().split("[#%]");  
 *try*{  
 Product.addProduct(productInfo[1], productInfo[2], productInfo[3], productInfo[4],  
 Integer.parseInt(productInfo[5]), Double.parseDouble(productInfo[6]));  
 } *catch*(IllegalArgumentException e){  
 *throw new* RuntimeException("Wrong type when add product)");  
 }  
 }  
 } *catch*(IOException e){  
 *throw new* RuntimeException(e);  
 }  
 }  
  
 */\*\*  
 \* Use at the end of program to write info to file  
 \* \*/  
 public static void* writeInfoToFile(){  
 writeGroups();  
 writeProducts();  
 }  
 *public static void* writeGroups(){  
 *try*{  
 StringBuilder result;  
 BufferedWriter bw = *new* BufferedWriter(*new* FileWriter(***groupList***));  
 *for*(Group group : Group.getGroups()){  
 result = *new* StringBuilder("#");  
 result.append(group.getGroupName());  
 result.append("%");  
 result.append(group.getGroupDescription());  
 result.append("\n");  
 bw.write(String.valueOf(result));  
 }  
 bw.close();  
 }*catch*(IOException e){  
 *throw new* RuntimeException(e);  
 }  
 }  
 *public static void* writeProducts(){  
 *try*{  
 StringBuilder result;  
 BufferedWriter bw = *new* BufferedWriter(*new* FileWriter(***productList***));  
 *for*(Product product: Product.getProducts()){  
 result = *new* StringBuilder("#");  
 result.append(product.getGroupNameInProduct());  
 result.append("%");  
 result.append(product.getProductName());  
 result.append("%");  
 result.append(product.getProductDescription());  
 result.append("%");  
 result.append(product.getProducer());  
 result.append("%");  
 result.append(product.getProductQuantityOnStock());  
 result.append("%");  
 result.append(product.getProductPrice());  
 result.append("\n");  
 bw.write(String.valueOf(result));  
 }  
 bw.close();  
 }*catch*(IOException e){  
 *throw new* RuntimeException(e);  
 }  
 }  
}

Group.java

*package* source;  
  
*import* java.util.ArrayList;  
  
*/\*\*  
 \* Існує декілька груп товарів (наприклад: Продовольчі, непродовольчі...).  
 \* Група товарів містить наступні властивості - назва, опис.\*/  
public class* Group {  
  
  
 *private* String **groupName**;  
 *private* String **groupDescription**;  
  
 *private static* ArrayList<Group> *groups* = *new* ArrayList<>();  
 *private static* ArrayList<Product> *products* = *new* ArrayList<>();  
  
 *private* Group(String groupName,String groupDescription){  
 *this*.**groupName**=groupName;  
 *this*.**groupDescription**=groupDescription;  
 }  
 *public static void* addGroup(String groupName, String groupDescription){  
 *boolean* unique = *true*;  
 *for*(Group group: getGroups()){  
 *if*(group.getGroupName().equals(groupName)){  
 unique = *false*;  
 *throw new* RuntimeException("Try to add non unique group)");  
 }  
 }  
 *if*(unique)  
 *groups*.add(*new* Group(groupName,groupDescription));  
 }  
 *public static void* editGroup(*int* index, String newGroupName, String newGroupDescription){  
 *boolean* unique = *true*;  
 *for*(Group group: getGroups()){  
 *if*(group.getGroupName().equals(newGroupName)){  
 unique = *false*;  
 *throw new* RuntimeException("Try to change group name for non unique");  
 }  
 }  
 *if*(unique){  
 *products* = Product.getProducts();  
 String oldName = *groups*.get(index).getGroupName();  
 *for* (Product product : *products*) {  
 *if* (product.getGroupNameInProduct().equals(oldName)) {  
 product.setGroupNameInProduct(newGroupName);  
 }  
 }  
 Product.setProducts(*products*);  
 *groups*.get(index).setGroupName(newGroupName);  
 }  
 *groups*.get(index).setGroupDescription(newGroupDescription);  
 }  
 *public static void* deleteGroup(String groupName){  
 *groups*.removeIf(group -> group.getGroupName().equalsIgnoreCase(groupName));  
 }  
 */\*\*Use when you remove group to remove all products in this group\*/  
 public static void* remove(String groupName){  
 *products* = Product.getProducts();  
 *for*(*int* i=0; i<*products*.size(); i++)  
 *if*(groupName.equals(*products*.get(i).getGroupNameInProduct()))  
 *products*.remove(i);  
 Product.setProducts(*products*);  
 }  
 *private static void* sortByName(){  
 Group current;  
 Group previous;  
 *for*(*int* i=0; i<*groups*.size(); i++){  
 *for*(*int* j=1;j< *groups*.size() - i; j++){  
 current = *groups*.get(j);  
 previous = *groups*.get(j-1);  
 *int* res = previous.getGroupName().toLowerCase().compareTo(current.getGroupName().toLowerCase());  
 *if*(res > 0){  
 *groups*.set(j-1,current);  
 *groups*.set(j,previous);  
 }  
 }  
 }  
 }  
 *public* String toString(){  
 *return* "\nGroup: "+getGroupName()+  
 "\nInfo: "+getGroupDescription();  
 }  
  
  
 */\*\*  
 \* block of getter and setter for fields  
 \* \*/  
  
 public static* ArrayList<Group> getGroups(){  
 sortByName();  
 *return groups*;  
 }  
 *public void* setGroups(ArrayList<Group> groups) {  
 *this*.*groups* = groups;  
 }  
  
 *public* String getGroupName() {*return* **groupName**;}  
 *public void* setGroupName(String groupName) {*this*.**groupName** = groupName;}  
  
 *public* String getGroupDescription() {*return* **groupDescription**;}  
 *public void* setGroupDescription(String groupDescription) {*this*.**groupDescription** = groupDescription;}  
}

MainFrame.java

*package* source;  
  
*import* javax.swing.\*;  
*import* javax.swing.border.EmptyBorder;  
*import* javax.swing.filechooser.FileNameExtensionFilter;  
*import* javax.swing.table.DefaultTableCellRenderer;  
*import* javax.swing.table.DefaultTableModel;  
*import* java.awt.\*;  
*import* java.awt.event.KeyAdapter;  
*import* java.awt.event.KeyEvent;  
*import* java.io.\*;  
*import* java.util.\*;  
*import* java.util.List;  
  
*public class* MainFrame *extends* JFrame {  
  
 *public static void* main(String[] args) {  
 MainFrame frame = *new* MainFrame();  
 frame.setVisible(*true*);  
 }  
  
 *public static final int* ***WIDTH*** = 700;  
 *public static final int* ***HEIGHT*** = 600;  
 *public static final int* ***BUTTON\_HEIGHT*** = 35;  
 *public static final int* ***LABEL\_HEIGHT*** = 25;  
 *public static final* Font ***PLAIN\_FONT\_14*** = *new* Font("Arial", Font.***PLAIN***, 14);  
 *public static final* Font ***PLAIN\_FONT\_16*** = *new* Font("Arial", Font.***PLAIN***, 16);  
 *public static final int* ***BIG\_INSERTS*** = 10;  
 *public static final int* ***SMALL\_INSERTS*** = 5;  
  
 GridBagConstraints **gbc**;  
  
 *private* JButton **saveButton**;  
 *private* JButton **cancelButton**;  
 *private* JButton **addButton**;  
 *private* JButton **openButton**;  
 *private* JButton **editButton**;  
 *private* JButton **deleteButton**;  
 *private* JButton **backButton**;  
 *private* JButton **writeOffButton**;  
 *private* JButton **searchButton**;  
 *private* JButton **printALlInfoButton**;  
 *private* JButton[] **buttons**;  
  
 *private* Map<String, JLabel> **labels**;  
  
 *private* JComboBox<String> **groupComboBox**;  
 *private* ButtonGroup **searchProperty**;  
 *private* JTextField **searchField**;  
 *private* JRadioButton **groupRadioButton**;  
 *private* JRadioButton **productRadioButton**;  
 *private* JTextField **nameField**;  
 *private* JTextArea **descriptionsTextArea**;  
 *private* JTextArea **allInfoTextArea**;  
 *private* JTextField **amountField**;  
 *private* JTextField **producerField**;  
 *private* JTextField **priceField**;  
  
 *private* JTable **groupsTable**;  
 *private* JTable **productsTable**;  
 *private* DefaultTableCellRenderer **centralRenderer**;  
  
 *private* State **current**;  
 *private* State **previous**;  
 *private int* **previousGroupIndex**;  
 *private int* **editIndex**;  
  
  
  
 *public* MainFrame(){  
 initFrame();  
 initComponents();  
 DataIO.readInfoFromFile();  
 getContentPane().add(getGroupsPanel());  
 initListeners();  
 **current** = State.***GROUPS***;  
 }  
  
 *private void* initComponents() {  
 **groupComboBox** = *new* JComboBox<>();  
 **groupComboBox**.setPreferredSize(*new* Dimension(-1, ***LABEL\_HEIGHT***));  
 **nameField** = *new* JTextField();  
 **nameField**.setName("Name Field");  
 **descriptionsTextArea** = *new* JTextArea();  
 **descriptionsTextArea**.setName("Descriptions Field");  
 **descriptionsTextArea**.setFont(***PLAIN\_FONT\_14***);  
 **allInfoTextArea** = *new* JTextArea();  
 **allInfoTextArea**.setFont(***PLAIN\_FONT\_14***);  
 **allInfoTextArea**.setBorder(*new* EmptyBorder(0, 0, 0, 0));  
 **allInfoTextArea**.setBackground(*this*.getBackground());  
 **allInfoTextArea**.setFocusable(*false*);  
 **amountField** = *new* JTextField();  
 **amountField**.setName("Amount Field");  
 acceptOnlyInteger(**amountField**);  
 **producerField** = *new* JTextField();  
 **producerField**.setName("Producer Field");  
 **priceField** = *new* JTextField();  
 **priceField**.setName("Price Field");  
 acceptOnlyDouble(**priceField**);  
 **searchField** = *new* JTextField();  
  
 **searchProperty** = *new* ButtonGroup();  
 **groupRadioButton** = *new* JRadioButton("Group");  
 **groupRadioButton**.setFont(***PLAIN\_FONT\_16***);  
 **productRadioButton** = *new* JRadioButton("Product");  
 **productRadioButton**.setFont(***PLAIN\_FONT\_16***);  
 **searchProperty**.add(**groupRadioButton**);  
 **searchProperty**.add(**productRadioButton**);  
  
 *for*(JComponent component: *new* JComponent[]{**nameField**, **amountField**, **producerField**, **priceField**, **searchField**}){  
 component.setFont(***PLAIN\_FONT\_14***);  
 component.setPreferredSize(*new* Dimension(-1, ***LABEL\_HEIGHT***));  
 }  
  
 initTableComponents();  
 initButtons();  
 initLabels();  
  
 **gbc** = *new* GridBagConstraints();  
 }  
  
 *private* JPanel getOpenAndEditGroupPanel() {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
 res.setBorder(*new* EmptyBorder(50, 50, 50, 50));  
 **nameField**.setText("");  
 **descriptionsTextArea**.setText("");  
 JScrollPane scrolledDescriptionsTextArea = *new* JScrollPane(**descriptionsTextArea**);  
 scrolledDescriptionsTextArea.setPreferredSize(*new* Dimension(-1, 200));  
 add(res, **labels**.get("Name"), **gbc**, 0, 0, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **nameField**, **gbc**, 1, 0, 1, 1,1, 1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Description"), **gbc**, 0, 1, 2, 1,1, 1, ***BIG\_INSERTS***);  
 add(res, scrolledDescriptionsTextArea, **gbc**, 0, 2, 2, 3, 1, 1, ***BIG\_INSERTS***);  
 add(res, **saveButton**, **gbc**, 0, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **cancelButton**, **gbc**, 1, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private void* fillOpenAndEditGroupPanel(*int* index) {  
 Group group = Group.getGroups().get(index);  
 **nameField**.setText(group.getGroupName());  
 **descriptionsTextArea**.setText(group.getGroupDescription());  
 }  
  
 *private* JPanel getSearchPanel() {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
  
 res.setBorder(*new* EmptyBorder(125, 75, 125, 75));  
  
 add(res, **labels**.get("Name"), **gbc**, 0, 0, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **searchField**, **gbc**, 1, 0, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Filter"), **gbc**, 0, 2, 1, 2, 1, 1, ***BIG\_INSERTS***);  
 add(res, **productRadioButton**, **gbc**, 1, 2, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **groupRadioButton**, **gbc**, 1, 3, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **searchButton**, **gbc**, 0, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **cancelButton**, **gbc**, 1, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private* JPanel getGroupsPanel() {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
  
 fillGroupTable();  
  
 JScrollPane scrolledGroupsTable = *new* JScrollPane(**groupsTable**);  
 scrolledGroupsTable.setPreferredSize(*new* Dimension(-1, 360));  
  
 add(res, scrolledGroupsTable, **gbc**, 0, 0, 3, 3, 1, 1, ***BIG\_INSERTS***);  
 add(res, **openButton**, **gbc**, 0, 3, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **editButton**, **gbc**, 1, 3, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **deleteButton**, **gbc**, 2, 3, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **addButton**, **gbc**, 0, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **searchButton**, **gbc**, 1, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **backButton**, **gbc**, 2, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **printALlInfoButton**, **gbc**, 1, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private void* fillGroupTable() {  
 DefaultTableModel model = (DefaultTableModel) **groupsTable**.getModel();  
 model.setRowCount(0);  
 *for* (Group group: Group.getGroups()) {  
 model.addRow(*new* String[] {group.getGroupName()});  
 }  
 **groupsTable**.setModel(model);  
 }  
  
 *private void* fillGroupTable(List<Group> groups) {  
 DefaultTableModel model = (DefaultTableModel) **groupsTable**.getModel();  
 model.setRowCount(0);  
 *for* (Group group: groups) {  
 model.addRow(*new* String[] {group.getGroupName()});  
 }  
 **groupsTable**.setModel(model);  
 }  
  
 *private void* fillProductTable(String group) {  
 DefaultTableModel model = (DefaultTableModel) **productsTable**.getModel();  
 model.setRowCount(0);  
 *try* {  
 *for* (Product product : ActionWithData.findProduct(group)) {  
 *if* (product.getGroupNameInProduct().equals(group))  
 model.addRow(*new* String[]{product.getProductName(), product.getProducer(),  
 String.valueOf(product.getProductQuantityOnStock()), String.valueOf(product.getProductPrice()),  
 String.valueOf(product.totalProductCost())});  
 }  
 }*catch* (RuntimeException e) {  
 System.***err***.println(e.getMessage());  
 }  
 **productsTable**.setModel(model);  
 }  
  
 *private void* fillProductTable(List<Product> products) {  
 DefaultTableModel model = (DefaultTableModel) **productsTable**.getModel();  
 model.setRowCount(0);  
 *for* (Product product: products) {  
 model.addRow(*new* String[] {product.getProductName(), product.getProducer(),  
 String.valueOf(product.getProductQuantityOnStock()), String.valueOf(product.getProductPrice()),  
 String.valueOf(product.totalProductCost())});  
 }  
 **productsTable**.setModel(model);  
 }  
  
  
  
 *private* JPanel getOpenAndEditProductPanel() {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
 **nameField**.setText("");  
 **descriptionsTextArea**.setText("");  
 **amountField**.setText("");  
 **priceField**.setText("");  
 **producerField**.setText("");  
 *for* (Group group :Group.getGroups()) **groupComboBox**.addItem(group.getGroupName());  
 res.setBorder(*new* EmptyBorder(0, 50, 0, 50));  
 JScrollPane scrolledDescriptionsTextArea = *new* JScrollPane(**descriptionsTextArea**);  
 scrolledDescriptionsTextArea.setPreferredSize(*new* Dimension(-1, 200));  
  
 add(res, **labels**.get("Group"), **gbc**, 0, 0, 1, 1, 0.1, 0.1, ***BIG\_INSERTS***);  
 add(res, **groupComboBox**, **gbc**, 1, 0, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Name"), **gbc**, 0, 1, 1, 1, 0.1, 0.1, ***BIG\_INSERTS***);  
 add(res, **nameField**, **gbc**, 1, 1, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Description"), **gbc**, 0, 2, 2, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, scrolledDescriptionsTextArea, **gbc**, 0, 3, 2, 3, 1, 1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Amount"), **gbc**, 0, 6, 1, 1, 0.1, 0.1, ***BIG\_INSERTS***);  
 add(res, **amountField**, **gbc**, 1, 6, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Cost"), **gbc**, 0, 7, 1, 1, 0.1, 0.1, ***BIG\_INSERTS***);  
 add(res, **priceField**, **gbc**, 1, 7, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **labels**.get("Producer"), **gbc**, 0, 8, 1, 1, 0.1, 0.1, ***BIG\_INSERTS***);  
 add(res, **producerField**, **gbc**, 1, 8, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **saveButton**, **gbc**, 0, 9, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **cancelButton**, **gbc**, 1, 9, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private void* fillOpenAndEditProductPanel(*int* index) {  
 Product product = Product.getProducts().get(index);  
 **groupComboBox**.setSelectedIndex(**previousGroupIndex**);  
 **nameField**.setText(product.getProductName());  
 **descriptionsTextArea**.setText(product.getProductDescription());  
 **amountField**.setText(String.valueOf(product.getProductQuantityOnStock()));  
 **priceField**.setText(String.valueOf(product.getProductPrice()));  
 **producerField**.setText(product.getProducer());  
 }  
  
 *private* JPanel getAllInfoPanel(){  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
 **allInfoTextArea**.setText(ActionWithData.getAllInfo());  
 JScrollPane scrolledAllInfoTextArea = *new* JScrollPane(**allInfoTextArea**);  
 scrolledAllInfoTextArea.setPreferredSize(*new* Dimension(-1, 400));  
 add(res, scrolledAllInfoTextArea, **gbc**, 0, 0, 2, 4, 1, 1, ***BIG\_INSERTS***);  
 add(res, **saveButton**, **gbc**, 0, 5, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 add(res, **backButton**, **gbc**, 1, 5, 1, 1, 1, 0.1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private* JPanel getProductsPanel(*int* groupIndex) {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
  
 fillProductTable(Group.getGroups().get(groupIndex).getGroupName());  
  
 JScrollPane scrolledGroupsTable = *new* JScrollPane(**productsTable**);  
 scrolledGroupsTable.setPreferredSize(*new* Dimension(-1, 380));  
 JLabel groupNameLabel = *new* JLabel("Group: " + Group.getGroups().get(groupIndex).getGroupName(), SwingConstants.***CENTER***);  
 groupNameLabel.setFont(***PLAIN\_FONT\_16***);  
  
 add(res, groupNameLabel, **gbc**, 0, 0, 3, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, scrolledGroupsTable, **gbc**, 0, 1, 3, 3, 1, 1, ***BIG\_INSERTS***);  
 add(res, **editButton**, **gbc**, 0, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **writeOffButton**, **gbc**, 1, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **deleteButton**, **gbc**, 2, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **addButton**, **gbc**, 0, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **searchButton**, **gbc**, 1, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **backButton**, **gbc**, 2, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private* JPanel getProductsPanel(List<Product> products) {  
 JPanel res = *new* JPanel(*new* GridBagLayout());  
  
 fillProductTable(products);  
  
 JScrollPane scrolledGroupsTable = *new* JScrollPane(**productsTable**);  
 scrolledGroupsTable.setPreferredSize(*new* Dimension(-1, 380));  
 JLabel groupNameLabel = *new* JLabel("", SwingConstants.***CENTER***);  
 groupNameLabel.setFont(***PLAIN\_FONT\_16***);  
  
 add(res, groupNameLabel, **gbc**, 0, 0, 3, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, scrolledGroupsTable, **gbc**, 0, 1, 3, 3, 1, 1, ***BIG\_INSERTS***);  
 add(res, **editButton**, **gbc**, 0, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **writeOffButton**, **gbc**, 1, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **deleteButton**, **gbc**, 2, 4, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **addButton**, **gbc**, 0, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **searchButton**, **gbc**, 1, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 add(res, **backButton**, **gbc**, 2, 5, 1, 1, 1, 1, ***BIG\_INSERTS***);  
 *return* res;  
 }  
  
 *private void* initTableComponents() {  
 **centralRenderer** = *new* DefaultTableCellRenderer();  
 **centralRenderer**.setHorizontalAlignment(SwingConstants.***CENTER***);  
  
 DefaultTableModel model = *new* DefaultTableModel();  
  
 model.addColumn("Name");  
 **groupsTable** = *new* JTable(model) {  
 *public boolean* editCellAt(*int* row, *int* column, java.util.EventObject e) {  
 *return false*;  
 }  
 };  
 **groupsTable**.getColumnModel().getColumn(0).setPreferredWidth((*int*) (0.6\****WIDTH***));  
 **groupsTable**.getColumnModel().getColumn(0).setResizable(*false*);  
  
 model = *new* DefaultTableModel();  
 model.addColumn("Name");  
 model.addColumn("Producer");  
 model.addColumn("Amount");  
 model.addColumn("Price");  
 model.addColumn("Overall Price");  
 **productsTable** = *new* JTable(model) {  
 *public boolean* editCellAt(*int* row, *int* column, java.util.EventObject e) {  
 *return false*;  
 }  
 };  
 **productsTable**.getColumnModel().getColumn(0).setPreferredWidth((*int*) (0.35\****WIDTH***));  
 **productsTable**.getColumnModel().getColumn(0).setResizable(*false*);  
 **productsTable**.getColumnModel().getColumn(1).setPreferredWidth((*int*) (0.35\****WIDTH***));  
 **productsTable**.getColumnModel().getColumn(1).setResizable(*false*);  
 **productsTable**.getColumnModel().getColumn(2).setPreferredWidth((*int*) (0.1\****WIDTH***));  
 **productsTable**.getColumnModel().getColumn(2).setResizable(*false*);  
 **productsTable**.getColumnModel().getColumn(3).setPreferredWidth((*int*) (0.1\****WIDTH***));  
 **productsTable**.getColumnModel().getColumn(3).setResizable(*false*);  
  
  
 **productsTable**.getColumnModel().getColumn(2).setCellRenderer(**centralRenderer**);  
 **productsTable**.getColumnModel().getColumn(3).setCellRenderer(**centralRenderer**);  
 **productsTable**.getColumnModel().getColumn(4).setCellRenderer(**centralRenderer**);  
  
 **groupsTable**.setRowHeight(25);  
 **productsTable**.setRowHeight(25);  
 **groupsTable**.setFont(***PLAIN\_FONT\_14***);  
 **productsTable**.setFont(***PLAIN\_FONT\_14***);  
 }  
  
  
 *private void* initButtons() {  
 **searchButton** = *new* JButton("Search");  
 **saveButton** = *new* JButton("Save");  
 **cancelButton** = *new* JButton("Cancel");  
 **addButton** = *new* JButton("Add");  
 **openButton** = *new* JButton("Open");  
 **editButton** = *new* JButton("Edit");  
 **deleteButton** = *new* JButton("Delete");  
 **backButton** = *new* JButton("Back");  
 **writeOffButton** = *new* JButton("Write Off");  
 **printALlInfoButton** = *new* JButton("Print All Info");  
 **buttons** = *new* JButton[]{**saveButton**, **cancelButton**, **addButton**, **openButton**, **editButton**, **deleteButton**, **backButton**, **writeOffButton**, **searchButton**, **printALlInfoButton**};  
 *for* (JButton button : **buttons**) {  
 button.setFont(***PLAIN\_FONT\_14***);  
 button.setPreferredSize(*new* Dimension(-1, ***BUTTON\_HEIGHT***));  
 }  
 }  
  
 *private void* initListeners(){  
 **openButton**.addActionListener(e -> {  
 *int* index = getSelectedRow();  
 *if*(index != -1) {  
 changePanel(getProductsPanel(index));  
 changeState(State.***PRODUCTS***);  
 }  
 **previousGroupIndex** = index;  
 });  
 **deleteButton**.addActionListener(e -> {  
 *int* index = getSelectedRow();  
 *if*(index != -1 && isConfirmation("Do you really want to delete " + ((**current** == State.***GROUPS***) ? **groupsTable**.getModel().getValueAt(index, 0) : **productsTable**.getModel().getValueAt(index, 0)) + "?")) {  
 *if*(**current** == State.***GROUPS***) {  
 Group.deleteGroup(Group.getGroups().get(index).getGroupName());  
 fillGroupTable();  
 }*else if* (**current** == State.***PRODUCTS***) {  
 Product.delete((String) **productsTable**.getModel().getValueAt(index, 0));  
 fillProductTable(Group.getGroups().get(**previousGroupIndex**).getGroupName());  
 }  
 }  
 });  
 **addButton**.addActionListener(e -> {  
 *if*(**current** == State.***GROUPS***) {  
 changePanel(getOpenAndEditGroupPanel());  
 }*else if* (**current** == State.***PRODUCTS***){  
 changePanel(getOpenAndEditProductPanel());  
 }  
 changeState(State.***ADDING***);  
 });  
 **backButton**.addActionListener(e -> {  
 *if*(**current** != State.***GROUPS***) {  
 changePanel(getGroupsPanel());  
 changeState(State.***GROUPS***);  
 }  
 });  
 **cancelButton**.addActionListener(e -> {  
 *if*(**current** != State.***GROUPS*** && **current** != State.***PRODUCTS***) {  
 *if*(**previous** == State.***GROUPS***) changePanel(getGroupsPanel());  
 *else if* (**previous** == State.***PRODUCTS***) changePanel(getProductsPanel(**previousGroupIndex**));  
 changeState(**previous**);  
 }  
 });  
 **searchButton**.addActionListener(e -> {  
 *if*(**current** != State.***SEARCHING***) {  
 changePanel(getSearchPanel());  
 changeState(State.***SEARCHING***);  
 } *else if* (**current** == State.***SEARCHING***) {  
 *if*(**groupRadioButton**.isSelected()) {  
 *try* {  
 ArrayList<Group> groups = ActionWithData.findGroup(**searchField**.getText());  
 changePanel(getGroupsPanel());  
 fillGroupTable(groups);  
 changeState(State.***GROUPS***);  
 }*catch* (RuntimeException ex) {  
 showMessage(ex.getMessage(), "Empty result!");  
 }  
 }*else if*(**productRadioButton**.isSelected()) {  
 ArrayList<Product> products;  
 *try* {  
 products = ActionWithData.findProduct(**searchField**.getText());  
 changePanel(getProductsPanel(products));  
 changeState(State.***PRODUCTS***);  
 }*catch* (RuntimeException ex) {  
 showMessage(ex.getMessage(), "Empty result!");  
 }  
 }  
  
 }  
 });  
 **editButton**.addActionListener(e -> {  
 *int* index = getSelectedRow();  
 *if*(index != -1) {  
 *if* (**current** == State.***GROUPS***) {  
 changePanel(getOpenAndEditGroupPanel());  
 fillOpenAndEditGroupPanel(index);  
 **editIndex** = index;  
 } *else if* (**current** == State.***PRODUCTS***) {  
 changePanel(getOpenAndEditProductPanel());  
 fillOpenAndEditProductPanel(index);  
 **editIndex** = index;  
 }  
 changeState(State.***OPENING\_AND\_EDITING***);  
 }  
 });  
 **saveButton**.addActionListener(e -> {  
 *if*(**current** == State.***OPENING\_AND\_EDITING***) {  
 *try* {  
 *if*(**previous** == State.***GROUPS***) {  
 checkIfIsNotEmpty(*new* JTextField[] {**nameField**});  
 checkIfIsNotEmpty(**descriptionsTextArea**);  
 Group.editGroup(**editIndex**, **nameField**.getText(), **descriptionsTextArea**.getText());  
 } *else if* (**previous** == State.***PRODUCTS***) {  
 checkIfIsNotEmpty(*new* JTextField[] {**nameField**, **producerField**, **amountField**, **priceField**});  
 checkIfIsNotEmpty(**descriptionsTextArea**);  
 Product.getProducts().get(**editIndex**).editProduct(Group.getGroups().get(**groupComboBox**.getSelectedIndex()).getGroupName(), **nameField**.getText(), **descriptionsTextArea**.getText(), **producerField**.getText(), Integer.parseInt(**amountField**.getText()), Double.parseDouble(**priceField**.getText()));  
 }  
 *if*(**previous** == State.***GROUPS***) changePanel(getGroupsPanel());  
 *else if* (**previous** == State.***PRODUCTS***) changePanel(getProductsPanel(**previousGroupIndex**));  
 changeState(**previous**);  
 }*catch* (RuntimeException ex) {  
 showMessage(ex.getMessage(), "Some trouble");  
 }  
 } *else if* (**current** == State.***ADDING***) {  
 *try* {  
 *if* (**previous** == State.***GROUPS***) {  
 checkIfIsNotEmpty(*new* JTextField[] {**nameField**});  
 checkIfIsNotEmpty(**descriptionsTextArea**);  
 Group.addGroup(**nameField**.getText(), **descriptionsTextArea**.getText());  
 } *else if* (**previous** == State.***PRODUCTS***) {  
 checkIfIsNotEmpty(*new* JTextField[] {**nameField**, **producerField**, **amountField**, **priceField**});  
 checkIfIsNotEmpty(**descriptionsTextArea**);  
 Product.addProduct(Group.getGroups().get(**groupComboBox**.getSelectedIndex()).getGroupName(), **nameField**.getText(), **descriptionsTextArea**.getText(), **producerField**.getText(), Integer.parseInt(**amountField**.getText()), Double.parseDouble(**priceField**.getText()));  
 }  
 *if*(**previous** == State.***GROUPS***) changePanel(getGroupsPanel());  
 *else if* (**previous** == State.***PRODUCTS***) changePanel(getProductsPanel(**previousGroupIndex**));  
 changeState(**previous**);  
 }*catch* (RuntimeException ex) {  
 showMessage(ex.getMessage(), "Some trouble");  
 }  
 } *else if* (**current** == State.***ALL\_INFO***) {  
 JFileChooser fileChooser = getFileChooser();  
 *if*(fileChooser.showSaveDialog(*this*) == JFileChooser.***APPROVE\_OPTION***) {  
 File file = fileChooser.getSelectedFile();  
 *try* (FileWriter writer = *new* FileWriter(file)){  
 writer.write(ActionWithData.getAllInfo());  
 } *catch* (IOException ex) {  
 *throw new* RuntimeException(ex);  
 }  
 }  
 *if*(**previous** == State.***GROUPS***) changePanel(getGroupsPanel());  
 *else if* (**previous** == State.***PRODUCTS***) changePanel(getProductsPanel(**previousGroupIndex**));  
 changeState(**previous**);  
 }  
 });  
 **writeOffButton**.addActionListener(e -> {  
 *int* index = getSelectedRow();  
 *if*(index != -1) {  
 String answer = JOptionPane.showInputDialog(*this*, "How much do you want to write " + "sth" + " off?", "", JOptionPane.***QUESTION\_MESSAGE***);  
 *if*(answer!= *null*) {  
 *if*(!isInteger(answer)) showMessage("You should enter a number!", "Invalid input!");  
 *else if*(Integer.parseInt(answer) <= 0) showMessage("You should enter a number that is larger than 0!", "Invalid input!");  
 *else* {  
 *try* {  
 Product.getProducts().get(index).writeOffFromStock(Product.getProducts().get(index).getProductName(), (Integer.parseInt(answer)));  
 fillProductTable(Group.getGroups().get(**previousGroupIndex**).getGroupName());  
 }*catch* (RuntimeException ex) {  
 showMessage(ex.getMessage(), "Invalid input!");  
 }  
 }  
 }  
 }  
 });  
 **printALlInfoButton**.addActionListener(e -> {  
 changePanel(getAllInfoPanel());  
 changeState(State.***ALL\_INFO***);  
 });  
 }  
  
 *private void* checkIfIsNotEmpty(JTextField[] textFields) {  
 *for*(JTextField textField: textFields){  
 *if* (textField.getText().isEmpty()) {  
 *throw new* RuntimeException("Fields '" + textField.getName() + "' is empty!\nYou should fill it out");  
 } *else if* (textField.getName().equals("Amount Field")) {  
 *if*(!isInteger(textField.getText()))  
 *throw new* RuntimeException("The value '" + textField.getText()+ "' is invalid!");  
 } *else if* (textField.getName().equals("Price Field")) {  
 *if*(!isDouble(textField.getText())) *throw new* RuntimeException("The value '" + textField.getText()+ "' is invalid!");  
 }  
 }  
 }  
  
 *private void* checkIfIsNotEmpty(JTextArea textArea) {  
 *if* (textArea.getText().isEmpty()) {  
 *throw new* RuntimeException("Fields '" + textArea.getName() + "' is empty!\nYou should fill it out");  
 }  
 }  
  
 *private void* showMessage(String text, String title) {  
 JOptionPane.showMessageDialog(*this*, text, title, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 *private* JFileChooser getFileChooser() {  
 JFileChooser fileChooser = *new* JFileChooser(".");  
 FileNameExtensionFilter filter = *new* FileNameExtensionFilter("TEXT FILES", "txt", "text");  
 fileChooser.setFileFilter(filter);  
 *return* fileChooser;  
 }  
  
 *private void* initLabels() {  
 **labels** = *new* HashMap<>();  
 **labels**.put("Name", *new* JLabel("Name:", SwingConstants.***CENTER***));  
 **labels**.put("Description", *new* JLabel("Description:", SwingConstants.***CENTER***));  
 **labels**.put("Group", *new* JLabel("Group:", SwingConstants.***CENTER***));  
 **labels**.put("Amount", *new* JLabel("Amount:", SwingConstants.***CENTER***));  
 **labels**.put("Producer", *new* JLabel("Producer:", SwingConstants.***CENTER***));  
 **labels**.put("Cost", *new* JLabel("Cost:", SwingConstants.***CENTER***));  
 **labels**.put("Filter", *new* JLabel("Filter:", SwingConstants.***CENTER***));  
 **labels**.values().forEach(l -> l.setFont(***PLAIN\_FONT\_16***));  
 }  
  
 *private void* initFrame() {  
 setTitle("Some Frame");  
 setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);  
 setBounds(100, 100, ***WIDTH***, ***HEIGHT***);  
 setMaximumSize(*new* Dimension((*int*) (1.1\****WIDTH***), (*int*) (1.1\****HEIGHT***)));  
 setMinimumSize(*new* Dimension((*int*) (0.9\****WIDTH***), ***HEIGHT***));  
 }  
  
 *public static void* add(JPanel panel, Component component, GridBagConstraints gbc, *int* x, *int* y, *int* width, *int* height, *double* wx, *double* wy, *int* insert) {  
 gbc.**insets** = *new* Insets(insert, insert, insert, insert);  
 gbc.**anchor** = GridBagConstraints.***CENTER***;  
 gbc.**fill** = GridBagConstraints.***HORIZONTAL***;  
 gbc.**gridx** = x;  
 gbc.**gridy** = y;  
 gbc.**gridwidth** = width;  
 gbc.**gridheight** = height;  
 gbc.**weightx** = wx;  
 gbc.**weighty** = wy;  
  
 panel.add(component, gbc);  
 }  
  
 *private void* changePanel(JPanel panel) {  
 getContentPane().removeAll();  
 getContentPane().add(panel);  
 *this*.revalidate();  
 *this*.repaint();  
 }  
  
 *private void* changeState(State newCurrentState) {  
 **previous** = **current**;  
 **current** = newCurrentState;  
 }  
  
 *private int* getSelectedRow() {  
 *int* index = -1;  
 JTable table = *null*;  
 *if*(**current** == State.***PRODUCTS***) table = **productsTable**;  
 *else if* (**current** == State.***GROUPS***) table = **groupsTable**;  
 *if* (table != *null*) index = table.getSelectedRow();  
 *if*(index == -1) {  
 *if* (table.getRowCount() > 0) table.setRowSelectionInterval(0, 0);  
 *else* JOptionPane.showMessageDialog(*this*, "The list is empty!");  
 }  
 *return* index;  
 }  
  
 *private boolean* isConfirmation(String text) {  
 *return* JOptionPane.showConfirmDialog(*this*, text, "Confirmation", JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_OPTION***;  
 }  
  
 *private boolean* isInteger(String answer) {  
 *try*{  
 Integer.parseInt(answer);  
 *return true*;  
 }*catch* (NumberFormatException e){  
 *return false*;  
 }  
 }  
  
 *private boolean* isDouble(String answer) {  
 *try*{  
 Double.parseDouble(answer);  
 *return true*;  
 }*catch* (NumberFormatException e){  
 *return false*;  
 }  
 }  
  
  
  
 *private void* acceptOnlyInteger(JTextField textField) {  
 textField.addKeyListener(*new* KeyAdapter() {  
 *public void* keyTyped(KeyEvent e) {  
 *char* c = e.getKeyChar();  
 *if* ( ((c < '0') || (c > '9')) && (c != KeyEvent.***VK\_BACK\_SPACE***)) {  
 e.consume();  
 }  
 }  
 });  
 }  
  
 *private void* acceptOnlyDouble(JTextField textField) {  
 textField.addKeyListener(*new* java.awt.event.KeyAdapter() {  
 *public void* keyTyped(KeyEvent e) {  
 *char* c = e.getKeyChar();  
 *if* (!((c >= '0') && (c <= '9') || (c == '.') || (c == KeyEvent.***VK\_BACK\_SPACE***) || (c == KeyEvent.***VK\_DELETE***))) {  
 getToolkit().beep();  
 e.consume();  
 }  
 }  
 });  
 }  
  
 *private enum* State {  
 ***GROUPS***, ***SEARCHING***, ***OPENING\_AND\_EDITING***, ***ADDING***, ***ALL\_INFO***, ***PRODUCTS*** }  
}

Product.java

*package* source;  
  
*import* java.util.ArrayList;  
  
*/\*\*  
 \* В кожній групі товарів існують конкретні товари (наприклад: борошно, гречка ...).  
 \* У кожного товару є наступні властивості - назва, опис, виробник, кількість на складі, ціна за одиницю.  
 \* \*/  
public class* Product {  
  
 *private* String **groupNameInProduct**;  
 *private* String **productName**;  
 *private* String **productDescription**;  
 *private* String **producer**;  
 *private int* **productQuantityOnStock**;  
 *private double* **productPrice**;  
  
 *private static* ArrayList<Product> *products* = *new* ArrayList<>();  
  
 *private* Product(String groupNameInProduct, String productName, String productDescription, String producer, *int* productQuantityOnStock, *double* productPrice) {  
 *this*.**groupNameInProduct**=groupNameInProduct;  
 *this*.**productName**=productName;  
 *this*.**productDescription**=productDescription;  
 *this*.**producer**=producer;  
 *this*.**productQuantityOnStock**=productQuantityOnStock;  
 *this*.**productPrice**=productPrice;  
 }  
 *public static void* addProduct(String groupNameInProduct, String productName, String productDescription, String producer, *int* productQuantityOnStock, *double* productPrice){  
 *boolean* unique = *true*;  
 *for*(Product product:*products*){  
 *if*(product.getProductName().equals(productName)){  
 unique = *false*;  
 *throw new* RuntimeException("Try to add non unique product");  
 }  
 }  
 *if*(unique && productPrice>0 && productQuantityOnStock>=0)  
 *products*.add(*new* Product(groupNameInProduct,productName,productDescription,producer,productQuantityOnStock,productPrice));  
 *else  
 throw new* RuntimeException("Try to add product with (lower than 0 price or negative quantity)");  
 }  
 *public void* editProduct(String groupNameInProduct, String productName, String productDescription, String producer, *int* productQuantityOnStock, *double* productPrice){  
 *boolean* uniqueName = *true*;  
 *boolean* groupExist = *false*;  
 *for*(Product product: *products*){  
 *if*(product.getProductName().equals(productName)){  
 uniqueName = *false*;  
 *throw new* RuntimeException("Try to change product name to non unique");  
 }  
 }  
 *if*(uniqueName)  
 setProductName(productName);  
 *for*(Group group: Group.getGroups()){  
 *if*(group.getGroupName().equals(groupNameInProduct)){  
 groupExist = *true*;  
 *break*;  
 }  
 }  
 *if*(groupExist)  
 setGroupNameInProduct(groupNameInProduct);  
 *else  
 throw new* RuntimeException("Try to change to non existing group");  
 *if*(productQuantityOnStock>0)  
 setProductQuantityOnStock(productQuantityOnStock);  
 *else  
 throw new* RuntimeException("Try to set incorrect quantity to product");  
 setProducer(producer);  
 setProductDescription(productDescription);  
 *if*(productPrice>0)  
 setProductPrice(productPrice);  
 *else  
 throw new* RuntimeException("Try to set incorrect price");  
  
 }  
 *public static void* delete(String name){  
 *products*.removeIf(p -> p.getProductName().equalsIgnoreCase(name));  
 }  
  
 */\*\*toString() in Product.class\*/  
 public* String toString(){  
 *return* "\n#"+**groupNameInProduct**+  
 "\nНазва: "+**productName**+  
 "\nОпис: "+**productDescription**+  
 "\nВиробник: "+**producer**+  
 "\nНа складі: "+**productQuantityOnStock**+  
 "\nЦіна за штуку: "+**productPrice**+  
 "\nЗагальна вартість: "+totalProductCost();  
 }  
 */\*\*  
 \* Method to add product on stock(change quantity)  
 \* \*/  
 public void* addToStock(String productNameToAdd,*int* quantity){  
 *for*(*int* i=0;i<getProducts().size();i++){  
 *if*(productNameToAdd.equals(*products*.get(i).getProductName())){  
 *products*.get(i).setProductQuantityOnStock(getProductQuantityOnStock()+quantity);  
 *break*;  
 }  
 }  
 }  
 */\*\*  
 \* Method to get product from stock(change quantity)  
 \* \*/  
 public void* writeOffFromStock(String productNameToWriteOff, *int* quantity){  
 *for*(*int* i=0;i<getProducts().size();i++){  
 *if*(productNameToWriteOff.equals(*products*.get(i).getProductName())){  
 *if*(*products*.get(i).getProductQuantityOnStock() >= quantity)  
 *products*.get(i).setProductQuantityOnStock(getProductQuantityOnStock()-quantity);  
 *else  
 throw new* RuntimeException("Try to write off "+quantity+" product(On stock:"+*products*.get(i).getProductQuantityOnStock()+")");  
 *break*;  
 }  
 }  
 }  
  
 */\*\*  
 \* For count total product cost(one specific product)  
 \* \*/  
 public double* totalProductCost(){  
 *return* getProductQuantityOnStock()\*getProductPrice();  
 }  
  
 */\*\*  
 \* For count total product value on stock(All products value)  
 \* \*/  
 public static double* valueOfProductsOnStock(){  
 *double* total = 0.0;  
 *for*(Product product: getProducts()){  
 total += product.getProductPrice()\*product.getProductQuantityOnStock();  
 }  
 *return* total;  
 }  
  
 *private static void* sortByName(){  
 Product current;  
 Product previous;  
 *for*(*int* i=0; i<*products*.size(); i++){  
 *for*(*int* j=1;j< *products*.size() - i; j++){  
 current = *products*.get(j);  
 previous = *products*.get(j-1);  
 *int* res = previous.getProductName().toLowerCase().compareTo(current.getProductName().toLowerCase());  
 *if*(res > 0){  
 *products*.set(j-1,current);  
 *products*.set(j,previous);  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* getters and setter to fields  
 \* \*/  
 public static* ArrayList<Product> getProducts(){  
 sortByName();  
 *return products*;  
 }  
 *public static void* setProducts(ArrayList<Product> newProducts){*products* = newProducts;}  
  
 *public* String getGroupNameInProduct() {*return* **groupNameInProduct**;}  
 *public void* setGroupNameInProduct(String groupNameInProduct) {*this*.**groupNameInProduct** = groupNameInProduct;}  
  
 *public* String getProductName() {*return* **productName**;}  
 *public void* setProductName(String productName) {*this*.**productName** = productName;}  
  
 *public* String getProductDescription() {*return* **productDescription**;}  
 *public void* setProductDescription(String productDescription) {*this*.**productDescription** = productDescription;}  
  
 *public* String getProducer() {*return* **producer**;}  
 *public void* setProducer(String producer) {*this*.**producer** = producer;}  
  
 *public int* getProductQuantityOnStock() {*return* **productQuantityOnStock**;}  
 *public void* setProductQuantityOnStock(*int* productQuantityOnStock) {*this*.**productQuantityOnStock** = productQuantityOnStock;}  
  
 *public double* getProductPrice() {*return* productPrice;}  
 *public void* setProductPrice(*double* productPrice) {*this*.productPrice = productPrice;}  
  
}