metin, yazı tipi, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

T.C.

SİVAS UNIVERSITY OF SCIENCE AND TECHNOLOGY

**PREPARED BY**

MEHMET YUSUF OCAK

ID: 220201015

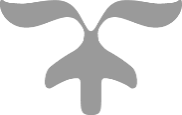
02 Ocak 2025

SİVAS



MODULE PROJECT

INVENTORY MANAGEMENT SYSTEM



02 Ocak 2025

SİVAS UNIVERSTY OF SCIENCE AND TECHNOLOGY

# Modul Courses:

Data Structere

Asst. Prof. Dr. Rezan BAKIR

Object-Orianted Programming

Asst. Prof. Dr. Zeshan IQBAL

# Contents

[Modul Courses: 2](#_Toc186308700)

[Contents 3](#_Toc186308701)

[PREFACE 5](#_Toc186308702)

[INTRODUCTION 6](#_Toc186308703)

[USAGE AREAS OF DATA STRUCTERE AND OOP 7](#_Toc186308704)

[1.Data Structures and Their Usage 7](#_Toc186308705)

[1.1 HashMap Usage 7](#_Toc186308706)

[1.2 ArrayList Usage 8](#_Toc186308707)

[1.3JTextArea and JScrollPane Usage 9](#_Toc186308708)

[2. OOP Approach 10](#_Toc186308709)

[2.1 Class Definitions 10](#_Toc186308710)

[2.1.1 Company Class 11](#_Toc186308711)

[2.1.2 Product Class 12](#_Toc186308712)

[2.1.3 Model Class 13](#_Toc186308713)

[2.1.4 InventoryApp Class 14](#_Toc186308714)

[3.Summary Table Of OOP Usage 15](#_Toc186308715)

[4.UML DIAGRAM 15](#_Toc186308716)

[JAVA CODE 16](#_Toc186308717)

[1.InventoryApp Class 16](#_Toc186308718)

[1.1Class Overview 16](#_Toc186308719)

[1.2 initUI() Method 17](#_Toc186308720)

[1.3 addProductDialog() Method 18](#_Toc186308721)

[1.4 addModelDialog() Method 18](#_Toc186308722)

[1.5 sellModelDialog() Method 19](#_Toc186308723)

[1.6 viewProductsDialog() Method 20](#_Toc186308724)

[1. 7 showBalance() Method 20](#_Toc186308725)

[2. Company Class 21](#_Toc186308726)

[2.1 Class Overview 21](#_Toc186308727)

[3.Product Class 22](#_Toc186308728)

[3.1 Class Overview 22](#_Toc186308729)

[4. Model Class 23](#_Toc186308730)

[4. 1 Class Overview 23](#_Toc186308731)

[5. Main Function 25](#_Toc186308732)

[6.Administrator Dashboard 26](#_Toc186308733)

[6.1 Button Login Action Performed 26](#_Toc186308734)

[6.2.Check Box Action Performed(Show Password Button): 26](#_Toc186308735)

[INTERFACE OF PROJECT 27](#_Toc186308736)

[1.ADMINISTRATOR DASHBOARD 27](#_Toc186308737)

[1.1 Code Flow: 27](#_Toc186308738)

[1.2 Scenarios: 27](#_Toc186308739)

[2.INVENTORY MANAGEMENT SYSTEM SCREEN 29](#_Toc186308740)

[2.1 Register New Product Line (addProductButton): 30](#_Toc186308741)

[2.2 Define Product Model (addModelButton): 31](#_Toc186308742)

[2.3 Process Model Sale (sellModelButton): 31](#_Toc186308743)

[2.4 View Inventory Portfolio (viewProductsButton) 34](#_Toc186308744)

[2.5 Financial Overview (balanceButton): 34](#_Toc186308745)

[CONCLUTION 35](#_Toc186308746)

[REFERANCE 36](#_Toc186308747)

# PREFACE

We would like to express our deepest gratitude to our honored professors, Asst. Prof. Dr. Rezan Bakır and Asst. Prof. Dr. Zeshan Iqbal, for their invaluable guidance and support throughout the development of this project. Their insights and experiences have enabled us to approach problems from every angle, significantly improving the quality of our research as a whole. We would also like to thank Hatice Aktaş, Research Assistant, and Emre Yüksek, Research Assistant, for their contributions and encouragement throughout the project.

In this project, we developed the Inventory Management System desktop application. This system shall provide ease in managing products and models, organize the process around sales, and further provide the user with means to analyze inventory information. The application is developed on Java programming language and the Graphical User Interface is highly interactive and user-friendly.

One of the biggest challenges during the development process was to create a framework that could manage the multi-tiered inventory data and raise the interaction by users. However, this challenge was overcome with the guidance of our professors: Asst. Prof. Dr. Rezan Bakır and Asst. Prof. Dr. Zeshan Iqbal.

In this new era of rapid technological advancement, the creation of innovative solutions in fields such as inventory management stands out. It is our hope that this project serves to help continuous technological advancements that make life easier and lay a foundation for future improvements.

Sadık Can GÜLER – Mehmet Yusuf OCAK

# INTRODUCTION

This project was designed to put into practice the basic concepts learned in Object-Oriented Programming (OOP) and Data Structures courses in a real-life setting. The application is based on the inventory management system of a company using OOP and proper data structures in Java that enable efficient storage, handling, and retrieval of products along with their respective models.

The whole project structure is based on these interdependent classes such as Company, Product, and Model. In this case, classes were designed with each class having a clear purpose so as to foster modularity and scalability. For instance, the Company class now holds many instances of the Product class; further, each Product class now handles many instances of Model in this hierarchical structure, created by efficient data structures—hash maps, for easy and effective data storage and retrieval.

The graphical user interface (GUI), developed using the Swing library, allows users to interact with these data structures with ease. Through the interface, users can perform operations such as adding products, modifying model specifications, and carrying out sales transactions. All the sales activities take into account several commercial factors like taxes and shipping costs, and detailed records of these transactions are maintained. This approach models business activities in the real world in a very organized way. This project tries to make good use of data structures to handle large-scale data and show the advantages of object-oriented design. It is user-friendly but also has a solid OOP foundation and an optimized mechanism of data management in the background.

# USAGE AREAS OF DATA STRUCTERE AND OOP

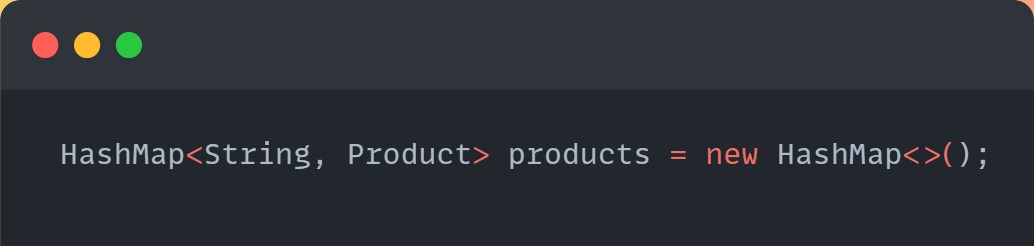
## 1.Data Structures and Their Usage

Various data structures are used in the code. Below are the data structures used and their detailed explanations along with their usage in the code.

### HashMap Usage

The HashMap data structure is used to store information in a form of key-value pairs, allowing fast access to the data. In this project, the Company class uses HashMap to keep track of the products. Every product is mapped to its unique name for quick identification. Similarly, the models attribute in the Product class stores information on product models, which are identified by their unique identifiers (id), allowing for fast access by these identifiers. One of the most valuable advantages of using a HashMap is its time complexity—represented as O(1)—for data retrieval; hence, it is especially suitable for use in inventory management systems that deal with large datasets. Moreover, it ensures data integrity by preventing the occurrence of duplicate entries through the specification of unique keys. The data structure in which this ideal balance between memory usage and operating efficiency is reached makes it one of the best choices for storing dynamic and complex sets of data. It provides structured control over products and models, reduces processing time due to fast access, and allows for possible scalability. These features have made HashMap extremely popular for dynamic high-performance data handling.

The HashMap data structure is used to dynamically manage products and models.



In this code, the Product objects are dynamically added and managed in the Company class.

Another example of its usage is in the Model class:

Here, the HashMap is used to provide quick access and update functionality for model details.

### ArrayList Usage

The ArrayList acts as a sequential and dynamic data structure, thus offering high flexibility in storing and accessing data. In this framework, the Model class makes use of ArrayList to handle the specifics of saleDetails. Such details are stored sequentially and hence easily accessed whenever needed. This dynamic resize ability of ArrayList makes it very apt for use in cases where the amount of data to be stored is unknown. For example, every time a model is sold, the related sales data are automatically added to the ArrayList. Since the elements can be accessed using their index, accessing a sequence of data occurs at high speed. Furthermore, this feature of maintaining the order of elements allows storing the sales data chronologically. In terms of time complexity, ArrayList has an amortized O(1) for adding an element and O(1) for accessing each element based on its index. Hence, this feature makes it a very good fit for handling dynamic data, which is updated constantly. Consequently, ArrayList would become the most appropriate choice for structuring and maintaining sales records in this system.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduArrayList is used to dynamically store sale details.

In this code, in the Model class, ArrayList is used to hold sale information. It provides flexibility for handling different types of sales details dynamically.

### 1.3 JTextArea and JScrollPane Usage

The components JTextArea and JScrollPane are used in the user interface to display textual information to the user in a graphical form. More precisely, JTextArea is a multi-line text box component and is used in the system in order to display the product catalog as well as information on sales to users. Thus, in the methods viewProductsDialog() and showBalance(), the detailed information of either products or total revenues is well displayed in a JTextArea. This component can take a large amount of text data and therefore provide easy access to information for users. However, in the case of large text content, JScrollPane is put into use to increase the usability of the overall application for the users. The JScrollPane adds scrollbars to the JTextArea to view more content without increasing the size of the text box. This amalgamation presents a visually appealing and operationally efficient user interface, ensuring that users can navigate information with ease, even in data-intensive environments. The JTextArea and JScrollPane components have provided the best solution for offering a user-centric design and functionality.



These structures are used for text inputs and managing user interface elements.

These data structures provide an efficient balance between data management and user interface in the code.

## 2. OOP Approach

In the Object-Oriented Programming (OOP) approach, the following key concepts have been utilized.

### 2.1 Class Definitions

The code defines the following classes:

* **Company:** Manages the collection of products and serves as the overarching structure for the inventory system.
* **Product:** Represents a single product with multiple models.
* **Model:** Represents a specific version of a product with attributes like price, stock, and revenue.
* **InventoryApp:** Serves as the main application class, handling the graphical user interface (GUI) and user interactions. This class inherits from JFrame to utilize its built-in GUI features.

### 2.1.1 Company Class

**Purpose:**

* Represents the business entity managing the inventory. Acts as a central hub for handling all products and their models.

**Key Responsibilities:**

* Maintains a collection of products.
* Provides operations to add products, list products, and retrieve specific product details

**Encapsulation**

* Definition: The Company class encapsulates a list of products, ensuring that only authorized methods can modify this list.
* Implementation:
* The private products attribute is accessed through methods such as addProduct, listProducts, and getProductByName.
* This ensures the integrity of the products list.

### 2.1.2 Product Class

**Purpose:**

* Represents an individual product in the inventory. Each product can contain multiple models.

**Key Responsibilities:**

* Stores the product name and its associated models.
* Manages operations related to the product, such as adding models or fetching details

**Encapsulation**

* Definition: Encapsulation ensures that the internal state of the Product class is protected from unauthorized access.
* Implementation:
* Private attributes (name, models) are accessed and modified only through getter and setter methods.
* This safeguards the integrity of the Product object while allowing controlled manipulation.

**Polymorphism**

* Definition: Polymorphism allows the Product class to override the toString method from the Object class, providing a custom string representation.
* Use Case: When Product objects are displayed, they use the customized toString method instead of the default implementation.

### 2.1.3 Model Class

**Purpose:**

* Represents an individual model associated with a product. It contains attributes such as name, price, and quantity.

**Key Responsibilities:**

* Stores the details of a specific product model.
* Provides methods to manipulate these details, such as updating quantity or calculating total price.

**Encapsulation**

* Definition: Private attributes store the internal state of the Model class (name, price, quantity). Getter and setter methods ensure controlled access and modification.
* Implementation:
* This prevents unauthorized or incorrect changes to the attributes, such as setting a negative price or quantity.

**Polymorphism**

* Definition: Like the Product class, the Model class also overrides the toString method to provide a customized description of the object.
* Use Case: When models are displayed, they use the overridden toString method to display relevant details.

### 2.1.4 InventoryApp Class

**Purpose:**

The InventoryApp class is the entry point of the application. It serves two key purposes:

* Manages the Graphical User Interface (GUI) for interacting with the user.
* Acts as the main controller that ties together the business logic implemented in the Company, Product, and Model classes.

**Key Responsibilities:**

* Handles user input via dialogs.
* Provides a menu of actions (e.g., adding products, selling models, viewing product details).
* Displays information such as product lists and sales balance in the GUI.

**Encapsulation**

* Definition: The InventoryApp class encapsulates the graphical user interface (GUI) logic and operations.
* Implementation:
* Private methods such as initUI modularize the GUI creation process.
* This ensures that the internal GUI setup is not exposed outside the class.

**Polymorphism**

* Definition: The InventoryApp class uses polymorphism when calling overridden toString methods from Product and Model classes.
* Use Case: This is evident in GUI components like displaying product and model details

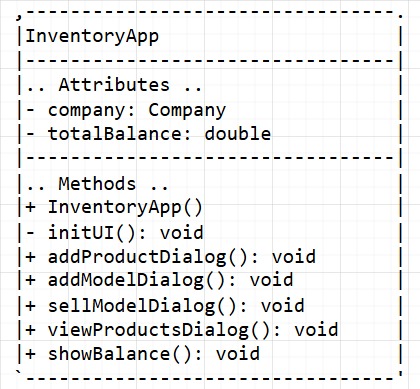
**Inheritance**

* Definition: The InventoryApp class extends JFrame, inheriting its attributes and methods to create a GUI application.
* Use Case: This allows the InventoryApp class to use JFrame’s capabilities, such as window creation and event handling.

## 3.Summary Table Of OOP Usage

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Inheritance** | **Polymorphism** | **Encapsulation** |
| Product | Not Used | toString override | Private attributes (name, models) |
| Model | Not Used | toString override | Private attributes (name, price) |
| Company | Not Used | Indirect (toString in listProducts) | Private products list |
| InventoryApp | Extends JFrame | Indirect (toString for products/models) | Private initUI method |

## 4.UML DIAGRAM



metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

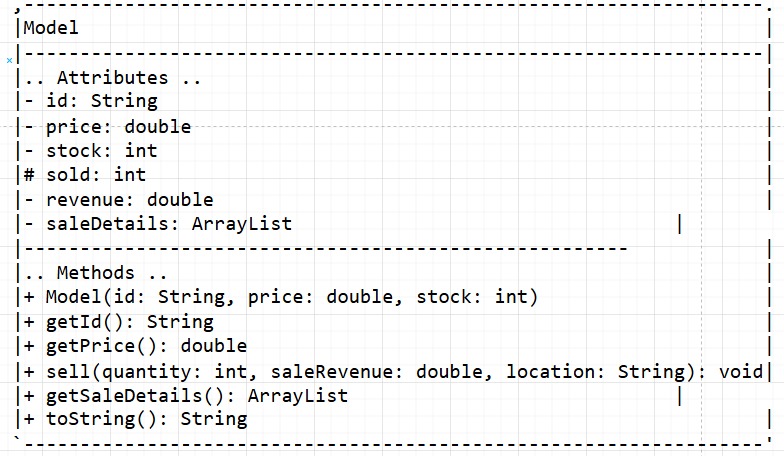
HAS A Relationship

(composition)

HAS A Relationship (aggregation)

metin, yazı tipi, sayı, numara, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu



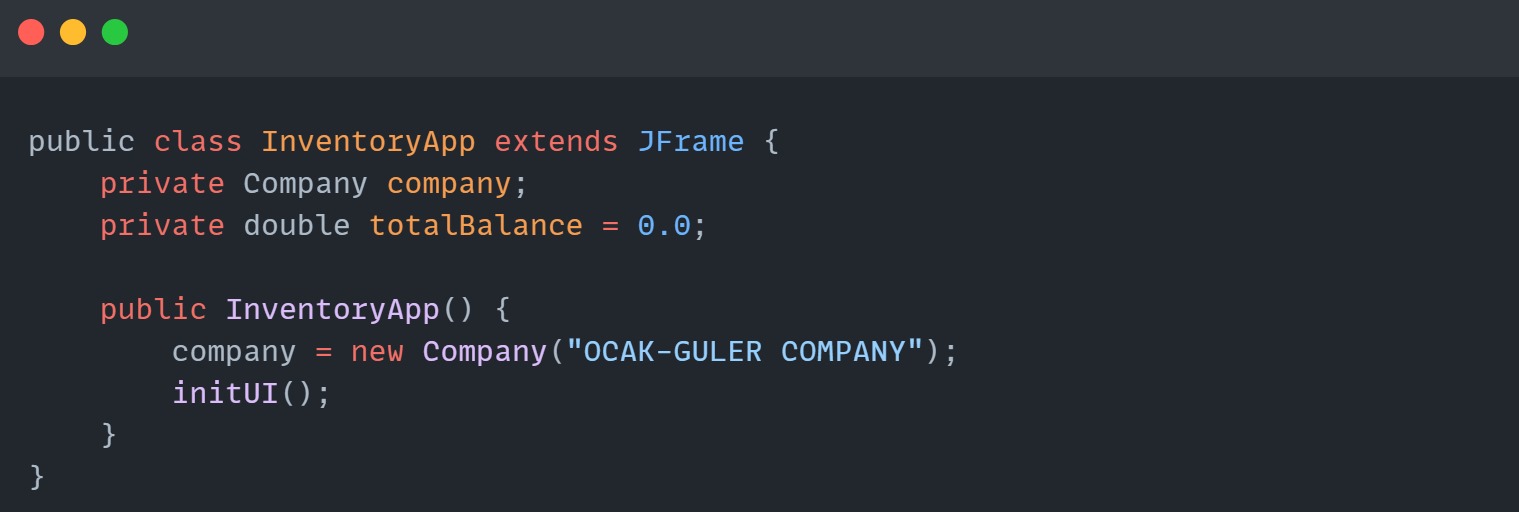
HAS A Relationship

(aggregation)

# JAVA CODE

## 1.InventoryApp Class

### 1.1Class Overview



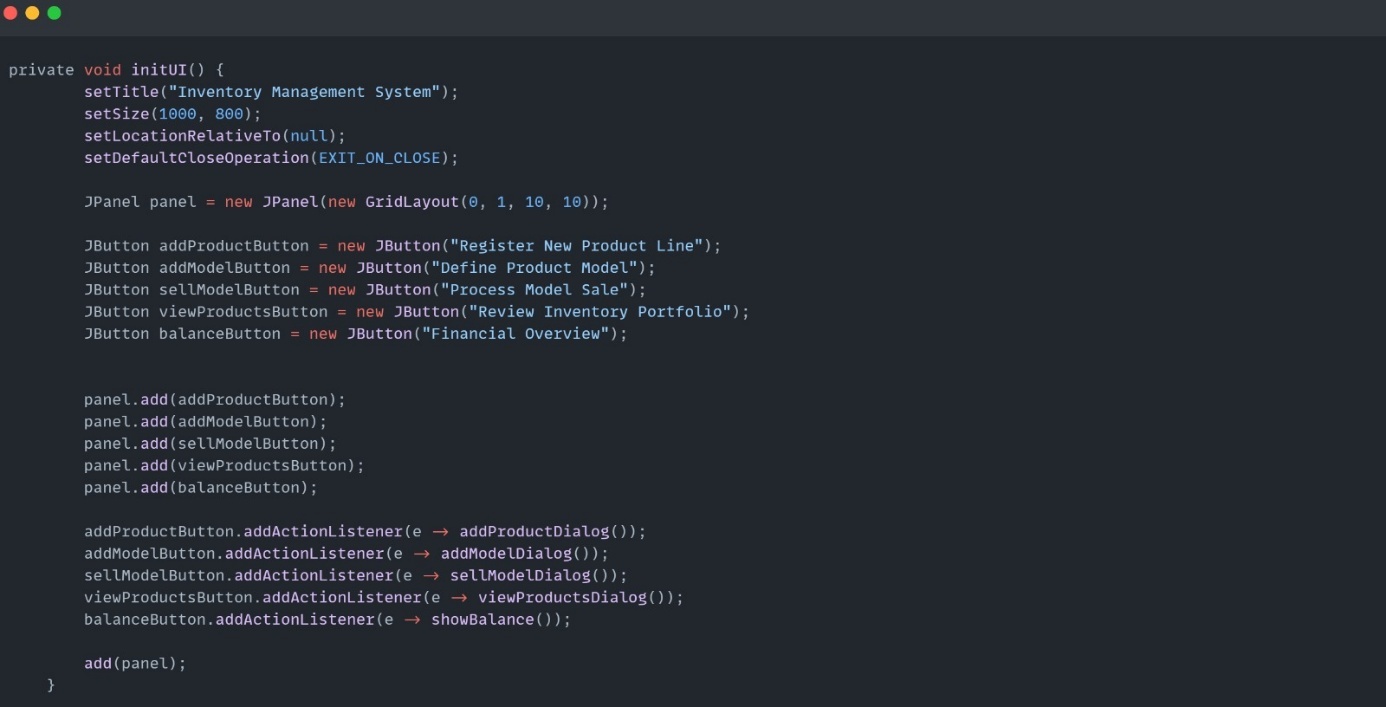
**Purpose:**

* This is the main class of the inventory management system, responsible for creating the user interface and linking it to the backend logic.

**Details:**

* company: A reference to the Company object, which manages the products.
* totalBalance: Tracks the total revenue generated from product sales.
* Constructor: Initializes the company and sets up the graphical user interface (GUI) using the initUI() method.

### 1.2 initUI() Method



**Purpose:**

* Sets up the main application window and adds buttons for managing the inventory system.

**Details:**

* Buttons: Each button corresponds to a specific function:
* Register New Product Line: Allows the user to add a new product to the inventory.
* Define Model Product: Adds a model (variant) to an existing product.
* Process Model Sale: Handles the sale of product models.
* Review Inventory Portfolio: Displays all products and their details.
* Financial Overview: Displays the total revenue and detailed sales records.
* ActionListeners: Each button is linked to a method that performs the corresponding action.

### 1.3 addProductDialog() Method



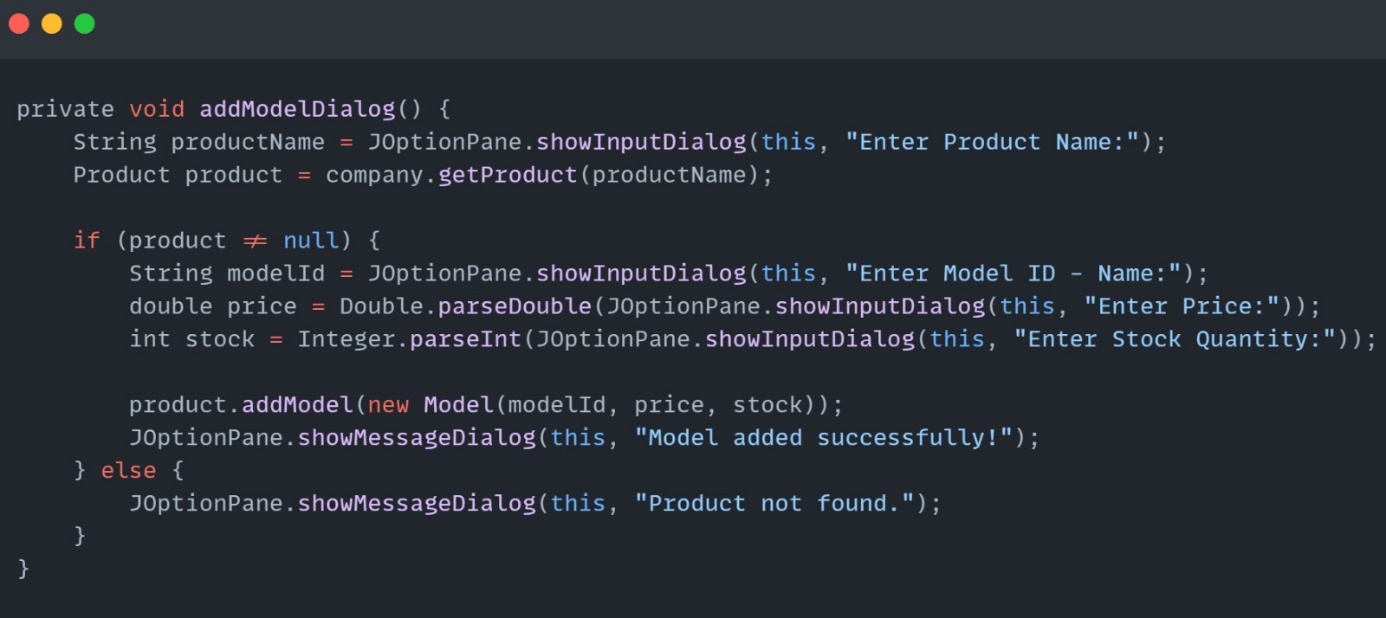
**Purpose**:

* Adds a new product to the company.

**Details:**

* Prompts the user to enter the product name using a dialog box.
* Creates a new Product object and adds it to the company.
* Shows a confirmation message upon success.

### 1.4 addModelDialog() Method



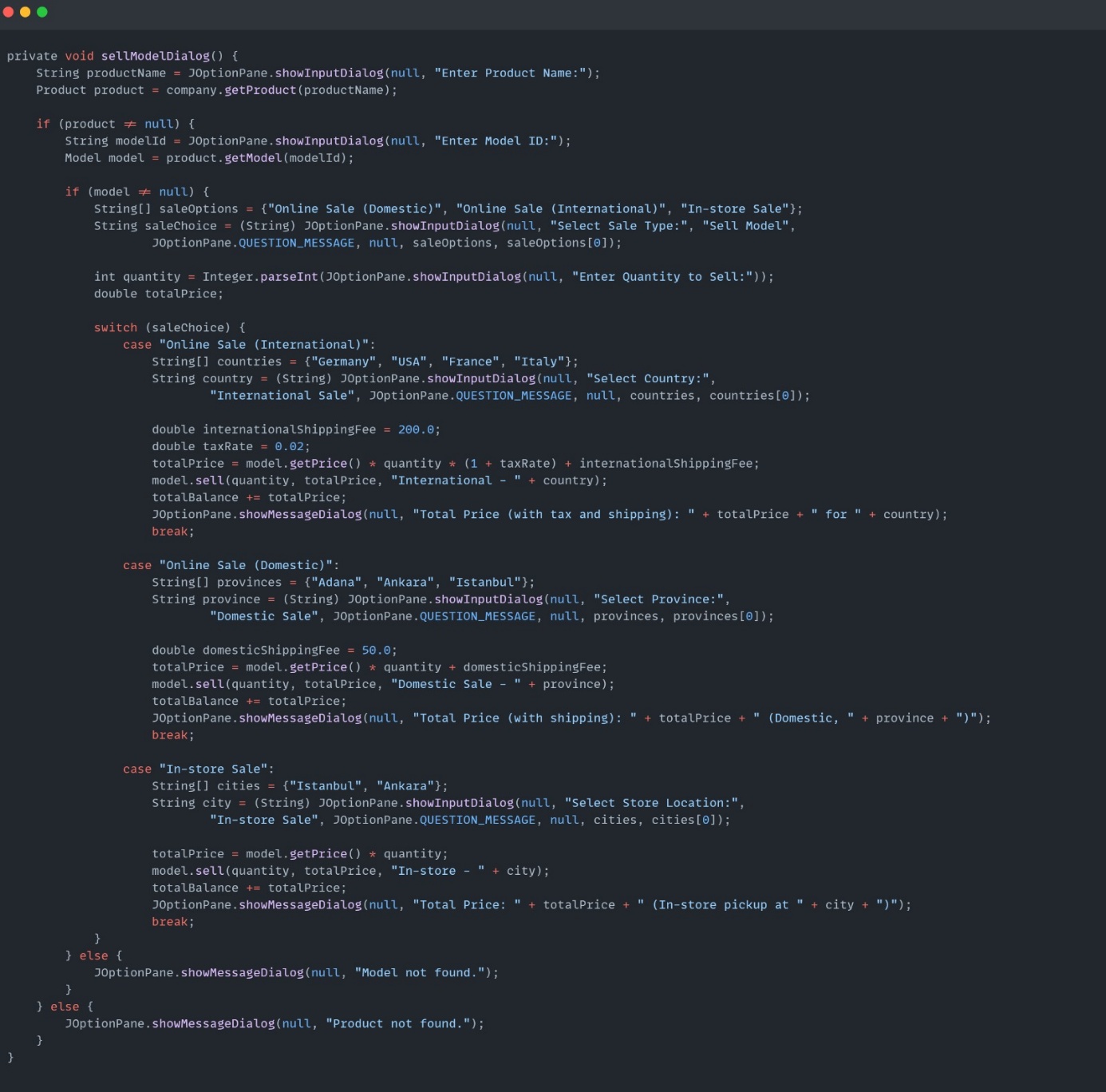
**Purpose**:

* Adds a model (variant) to an existing product.

**Details:**

* Finds the product by name.
* Prompts the user for model details (ID, price, stock quantity).
* Adds the new model to the product.
* Notifies the user of success or failure.

### 1.5 sellModelDialog() Method

****

**Purpose:**

* Manages product sales with different sale types (online domestic, international, or in-store).
* Incorporates location-based tax and shipping fee calculations for accurate pricing.

**Details:**

**Tax Rates:**

* Applied only for international online sales at a rate of 2% (0.02) on the total product price.
* No tax is applied for domestic or in-store sales.

**Shipping Fees**:

* International Online Sales: A fixed fee of 200 units is added to the total price.
* Domestic Online Sales: A fixed fee of 50 units is added to the total price.
* In-store Sales: No shipping fee is applied.

**Allows the user to:**

* Choose a sale type and location (e.g., country, province, or store).
* Input the quantity for the sale.

**Automatically:**

* Calculates the total price based on quantity, product price, applicable tax, and shipping fees.
* Updates the stock quantity and revenue details for the model.
* Displays a summary including the total price, tax, and shipping fee details to the user.

### viewProductsDialog() Method



**Purpose:**

* Displays all products and their models in a scrollable dialog.

**Details:**

* Uses JTextArea to format the product list and JScrollPane for scrolling.

### 1.7 showBalance() Method

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Purpose:**

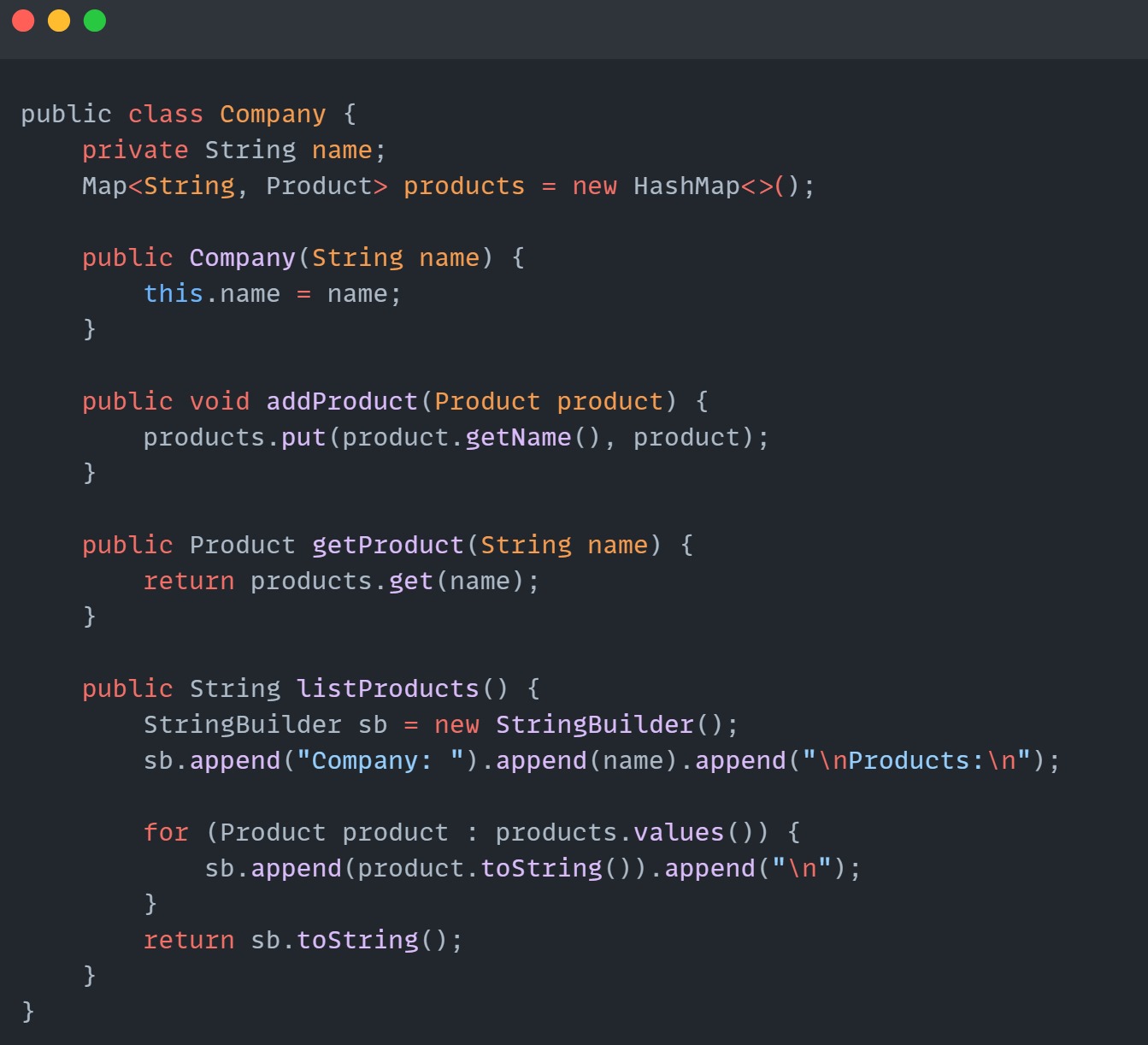
* Summarizes the total balance and all sales details.

**Details:**

* Aggregates sales from all products and models.
* Displays a detailed summary in a scrollable dialog.

## 2. Company Class

### 2.1 Class Overview



**Purpose:**

Represents the company and serves as a container for all products.

**Details:**

* products: A map where the keys are product names, and the values are the corresponding Product objects.

**Methods:**

* addProduct(Product product): Adds a product to the products map using its name as the key.
* getProduct(String name): Retrieves a product from the products map by name.
* listProducts(): Returns a formatted string containing the company name and all product details.

## 3.Product Class

### 3.1 Class Overview



**Purpose**:

* Represents an individual product and manages its models.

**Details:**

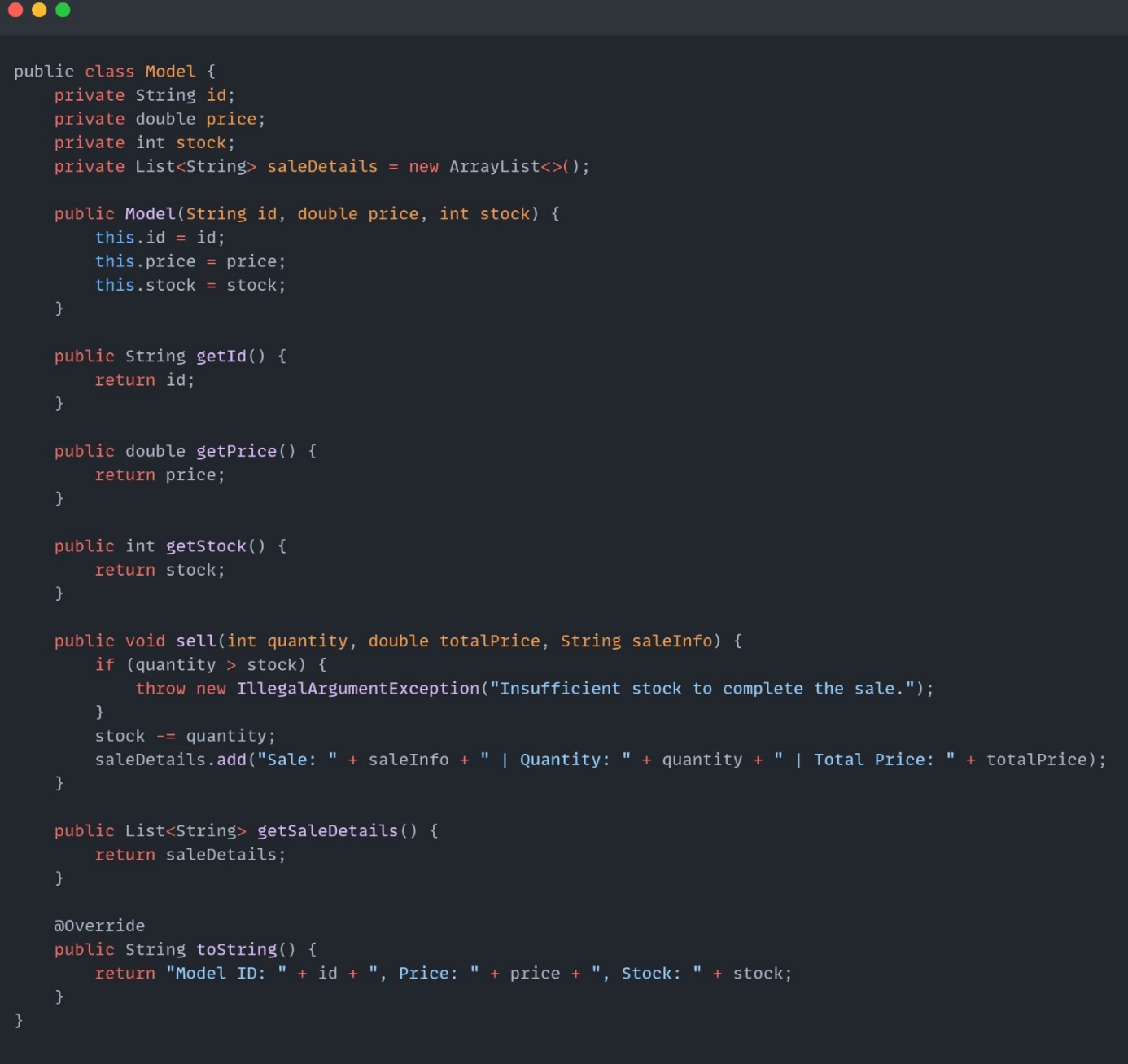
* models: A map where keys are model IDs, and values are Model objects associated with this product.

**Methods:**

* getName(): Returns the product name.
* addModel(Model model): Adds a new model to the models map.
* getModel(String id): Retrieves a model by its ID from the models map.
* toString(): Formats the product's details, including its name and associated mod

## 4. Model Class

### 4. 1 Class Overview



**Purpose:**

* Represents an individual product variant or model.

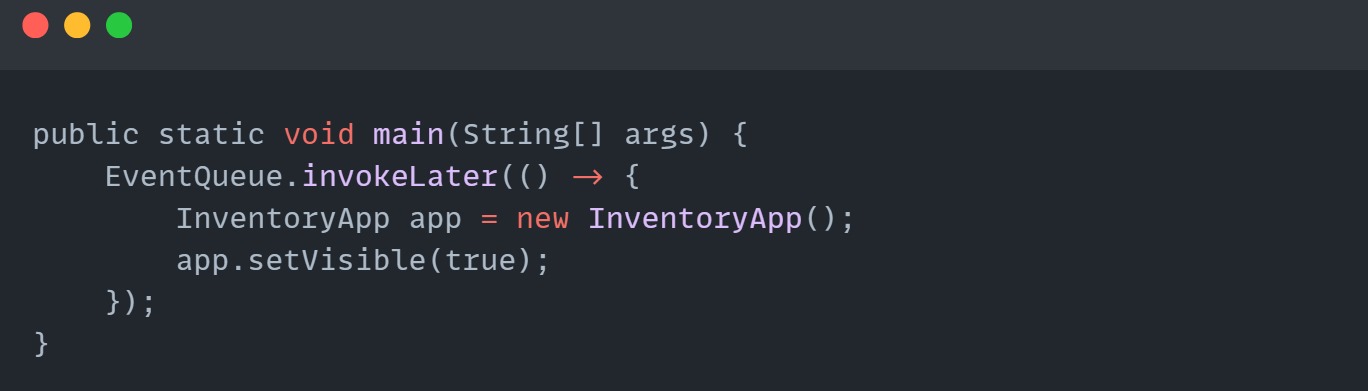
**Details:**

* id: A unique identifier for the model (e.g., "ID-01").
* price: The price of the model.
* stock: The available stock quantity for the model.
* saleDetails: A list of strings detailing the sales made for this model.
* Constructor: Initializes the model's ID, price, and stock.

**Methods:**

* getId(): Returns the model's ID.
* getPrice(): Returns the model's price.
* getStock(): Returns the available stock.
* sell(int quantity, double totalPrice, String saleInfo): Handles a sale by:
* Validating stock availability.
* Deducting the sold quantity from stock.
* Recording the sale details (type, quantity, total price).
* getSaleDetails(): Returns the list of sales made for this model.
* toString(): Returns a formatted string of the model's details.

## 5. Main Function

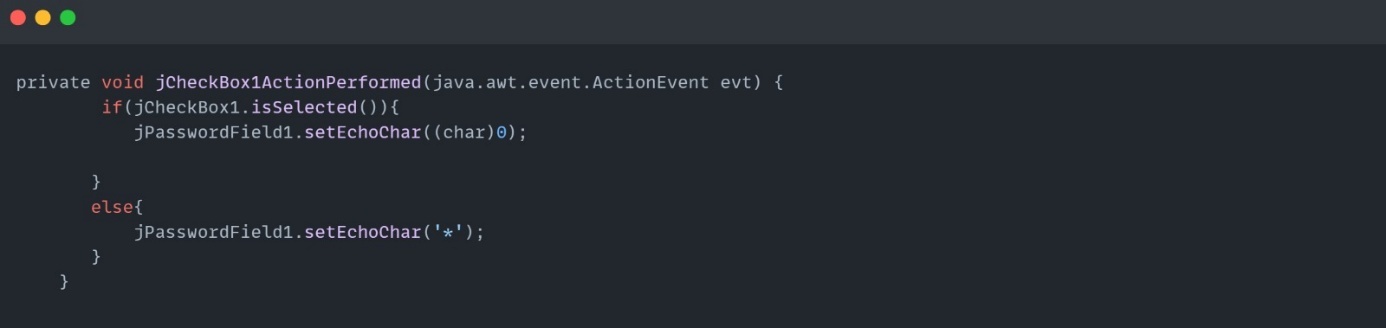


* The main function is the starting point of the program. Its purpose is to initialize and display the InventoryApp graphical user interface (GUI). It uses EventQueue.invokeLater to ensure the GUI runs on the Event Dispatch Thread, which is required for thread-safe operations in Swing.
* Creates an instance of InventoryApp.
* Sets the window visible using app.setVisible(true) to display the GUI.
* This connects the user to the inventory management system, allowing them to interact with the app through the provided buttons.

## 6.Administrator Dashboard

### 6.1 Button Login Action Performed

### 6.2.Check Box Action Performed(Show Password Button):



**Purpose**

This code gives functionality to a password field, jPasswordField1, to show/hide the password input when a checkbox, jCheckBox1, is checked/unchecked. If the checkbox is checked, the password field shows the password; otherwise, it masks the password using asterisks (\*).

**Detail**

**Selection of Checkbox**

* jCheckBox1 is used to check/uncheck the visibility of the password.
* The isSelected() method identifies whether the checkbox is checked or not.
* Password Visibility

**If the checkbox is checked:**

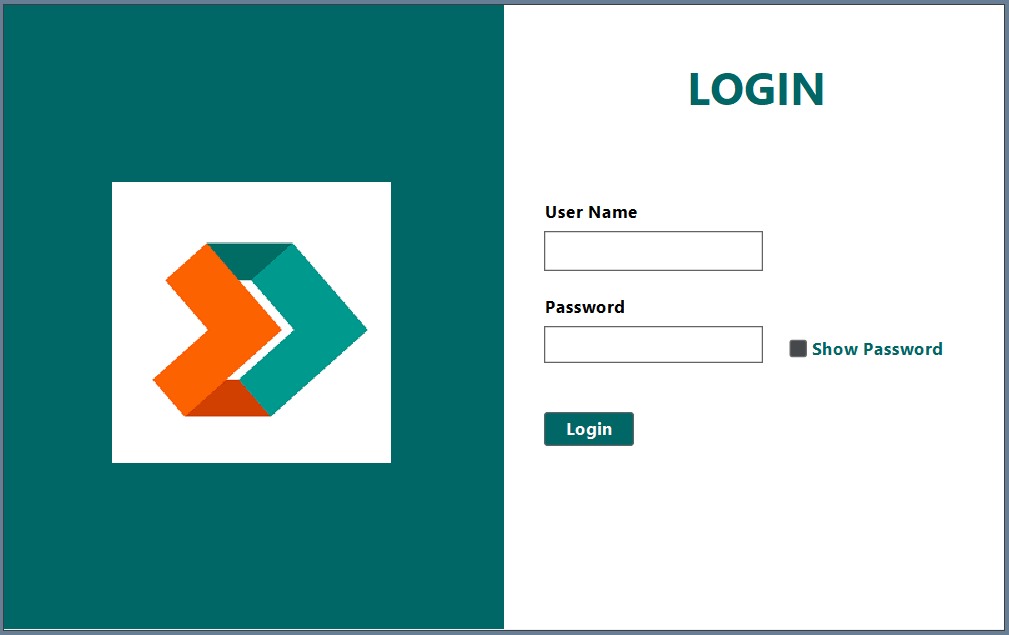
* Then jPasswordField1 fires the setEchoChar((char)0) method.
* This removes the mask characters. The password can now be viewed.

**If NOT checked :**

* The setEchoChar('\*') method is called upon jPasswordField1.
* The password is now masked with asterisks (\*).

# INTERFACE OF PROJECT

## 1.ADMINISTRATOR DASHBOARD



### Code Flow:

When the user enters the username and password and clicks the login button, the button\_loginActionPerformed function is triggered.

The code compares the entered information. If a match is found, a new window opens; otherwise, an error message is shown.

### Scenarios:

**Scenario 1:** If the username or password is incorrect:

"Login Failed !! Username or Password Incorrect" message is shown.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

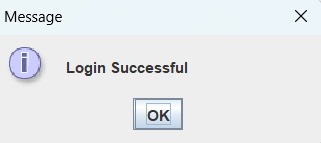
**Scenario 2:**

If the username is "sadikcan.guler" and the password is "271022":

A successful login message is shown, and InventoryApp is opened.

**Scenario 3**: If the username is "mehmetyusuf.ocak" and the password is "261119":

A successful login message is shown, and InventoryApp is opened.

****

In the event of a successful administrator entry, the inventory management system screen is displayed.

## 2.INVENTORY MANAGEMENT SYSTEM SCREEN

metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Register New Product Line (addProductButton):**

This button is responsible for allowing the user to register a new product line in the system. It opens a dialog that requests from the user a name for the product. If valid, a new Product is created and added to the company's product catalog.

**Define Product Model (addModelButton):**

This is the button to create a model for an existing product. A dialogue window opens to ask the user for the name of the product. If that product already exists, one can add a model providing ID, price, and quantity in stock. The model gets added to the list of models for that product.

**Process Model Sale (sellModelButton):**

This button enables the user to process a sale of some model present in the inventory. After choosing a product and model, the user has to choose one of the three options for the sale: Online Sale Domestic, Online Sale International, or In-store Sale. Each option will calculate the total sale price, adding shipping fees and taxes depending on the kind of sale. The model stock is updated, and the revenue from the sale is recorded.

**View Inventory Portfolio (viewProductsButton)**

This button opens the entire product catalog in a read-only view, listing all the products and their models, but not limited to model ID, price, stock, and units sold within a scrollable text area inside a message dialog.

**Financial Overview (balanceButton):**

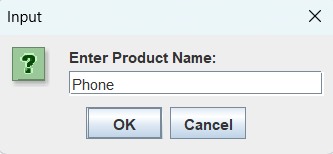
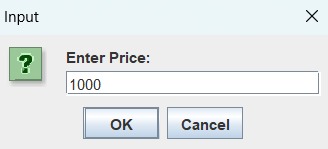
This button provides an overview of the company's total balance and detailed sales information. It shows the total revenues from the sales of all products and models, including each individual sale quantity, location, and price for each model. It helps the user to track the company's financial performance.

### 2.1 Register New Product Line (addProductButton):metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim Açıklama otomatik olarak oluşturuldu

Following the addition of the product, the message 'Product added successfully' is displayed.

### 2.2 Define Product Model (addModelButton):

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

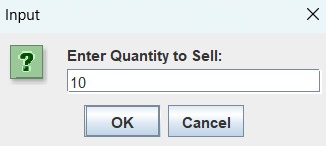
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

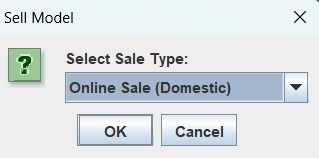
Açıklama otomatik olarak oluşturuldu

Following the determination of the product to which the model will be added, the price and stock quantity of said product are then ascertained. Subsequent to the execution of these operations, the message 'Model added successfully' is displayed on the screen.

### 2.3 Process Model Sale (sellModelButton):metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim Açıklama otomatik olarak oluşturuldu



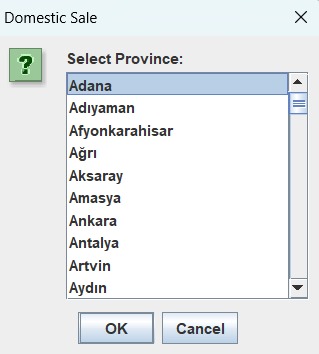


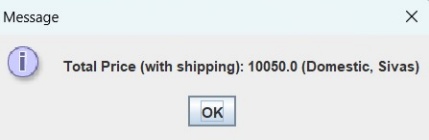


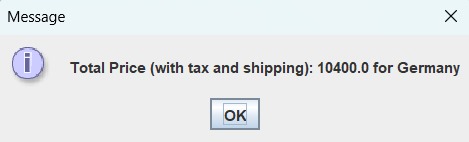
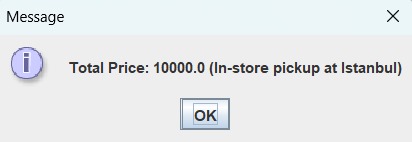
metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu







**Step 1**: **Entering Product and Model Information**

Product Name: The customer first enters the generic name of the product they wish to purchase (for example, “Phone”).

Model ID-Name: Next, the customer selects the specific model of the product (for example, “Samsung”).

Quantity to Sell: Finally, he/she indicates how many units of the product he/she wants to buy.

**Step 2: Selecting the Sales Type**

Sale Type: The system offers different sales types to the customer:

Online Sales (Domestic): Selling the product online within the country

Online Sales (International): Selling the product online to another country

In-Store Sale: Purchase of the product from a physical store

**Step 3: Determination of Details**

Local Sale: The customer chooses the province where the product will be shipped.

International Sales: The customer chooses the country where the product will be shipped.

Store Sales: The customer chooses the store where they want to buy the product.

**Step 4: Displaying the Total Price**

Based on the selected sale type and quantity, the system calculates the total price and presents it to the customer. This price may include additional charges such as taxes and shipping costs.

### 2.4 View Inventory Portfolio (viewProductsButton)

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

This tab presents data regarding the products and models that have been sold, in addition to the current inventory levels in the company warehouse.Furthermore, the total revenue derived from sales is exhibited.

### 2.5 Financial Overview (balanceButton):

metin, ekran görüntüsü, ekran, görüntüleme, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

This tab shows the breakdown of products sold (where they were sold to) and the total revenue generated from those sales. It also presents the net profit/income after subtracting related expenses such as shipping charges and taxes.

# CONCLUTION

It develops the practical implementation of OOP and data structures in Java by developing an Inventory Management System. The project showcases modularity, scalability, and user interaction through the use of a well-developed GUI using the Swing library. Key concepts of OOP, such as encapsulation, inheritance, and polymorphism, have been applied effectively in showcasing how OOP principles are superior in solving real-world problems.

The application utilizes efficient data management by using HashMap and ArrayList data structures to operate, such as adding of products, model management, sales handling, and more. Problems during the development process like making a dynamic framework that manages multi-tier inventories were possible to overcome through guidance and persistence.

This project fulfills the academic requirements, but it also forms the very foundation on which future enhancements to inventory management systems can be built. It showcases how the integration of programming knowledge and business requirements can lead to practical solutions.

# REFERANCE

* <https://www.geeksforgeeks.org/arraylist-in-java/>
* <https://www.geeksforgeeks.org/arraylist-in-java/>
* <https://www.youtube.com/watch?v=jHSBrX8lLWk&t=694s>
* <https://docs.oracle.com/javase/tutorial/getStarted/index.html>
* <https://chatgpt.com/>
* <https://www.geeksforgeeks.org/data-structures/>
* <https://www.w3schools.com/java/java_polymorphism.asp>
* <https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html>
* <https://www.guru99.com/java-swing-gui.html>
* <https://www.geeksforgeeks.org/java-awt-tutorial/>