**Astana IT University**

**PROJECT REPORT. ELECTRONIC MARKETPLACE.**

Object-Oriented Programming

Nurtas Nadirov

Almas Baisultan

Aruzhan Kuanova, Yernar Asherbekov, Ulan Abdramanov, Alikhan Olzhabaev   
  
 **SE-2227**

**REPORT CONTENT**   
**1.INTRODUCTION**

**2.OVERVIEW**   
**3.THE PROJECT PART**

**4.RESPONSIBILITIES**

**5.SINGLETON PATTERN**  
**6.REVIEW**

**7.CONCLUSION**   
**8.REFERENCE LIST**

**INTRODUCTION**

**Formulation of the problem**

Create marketplace with one of product or service (food, clothes, electronics, cars and etc.) with 5 hierarhium steps and ~150-200 objects.

Application development requirements

**The main criterias of assignment:**

**⦁** JDBC,

⦁ Database connection,

⦁ S.O.L.I.D.

**Also, use the capabilities of OOP:**

**⦁** classes,

⦁ inheritance,

⦁ polymorphism,

⦁ encapsulation.

This project represents a simple market application written in Java, which connects to a PostgreSQL database using JDBC driver. The program allows users to add their information, view the available products in different categories, and make purchases.

**Here is a brief overview of the functions and features of the code:**

The program starts by defining the necessary dependencies, including the JDBC driver, the SQL package, and the Scanner class for user input. It also sets up the database connection parameters, including the database URL, the username, and the password.

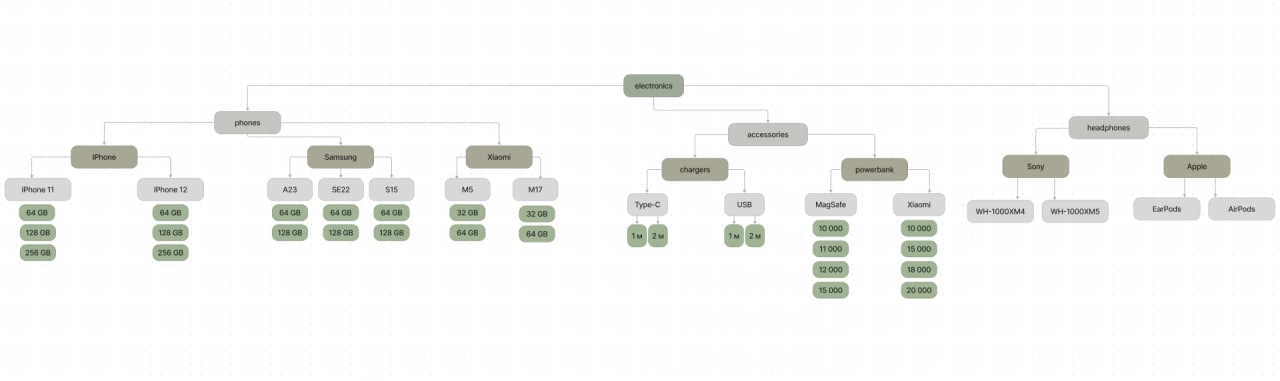
The add\_user() function allows users to register their names and balances in the database. It prompts the user to enter their name and balance, then uses a prepared statement to insert the user's information into the "users" table. It then retrieves the newly assigned user ID from the database, which is used to identify the user for subsequent transactions.

The show\_catalog() function displays the available product categories and prompts the user to choose a category. Depending on the user's choice, it calls the corresponding show\_phones(), show\_tablets(), or show\_accessories() functions to display the products in that category. The user can exit the category selection by choosing option 4.

The show\_phones(), show\_tablets(), and show\_accessories() functions retrieve the relevant product information from the database using SQL queries, then print the results to the console.

The purchase() function prompts the user to choose a product category and displays the available products in that category using the corresponding show\_\*() function. It then prompts the user to enter the product ID they wish to purchase and retrieves the price of that product from the database. If the user's balance is sufficient, it subtracts the product price from their balance and updates their balance in the database. If the user's balance is insufficient, the purchase is not completed and an error message is displayed.

Overall, this code represents a simple command-line market application that uses a PostgreSQL database to store product and user information. While the application is functional, there are several areas for improvement, such as adding error handling, input validation, and exception handling to make the program more robust and user-friendly.

**PROJECT PART**  
**This is our plan or a figma picture that shows the hierarchy of our project.**  
Created by Alikhan

**RESPONSIBILITIES**   
  
**Name Contribution to the project**

|  |  |
| --- | --- |
| Aruzhan Kuanova | Team lead, model, menu, creator, project structure |
| Ernar Asherbekov | Classes, 150-200 objects, helper |
| Ulan Abdramanov | Second creator, menu, users and purchasing |
| Alikhan Olzhabaev | Created figma, helper |

**Singleton pattern**

We have decided to use singleton pattern for our project since it has following advantages

|  |  |
| --- | --- |
| **Global Access Point** | A Singleton is a way to safeguard global variables by disguising them as a global object with a single access point. Although accessible from anywhere in the program, a Singleton cannot be modified from outside and can only be changed internally. Essentially, it serves as a method to protect global variables. |
| **Unique Entity Model** | When you model real-world entities in your program, it becomes simpler to comprehend and analyze. Real-world scenarios often involve Singletons, such as registration offices, global timers, or unique ID factories. As a result, by incorporating such Singletons into your program, you can achieve a great alignment between the program's abstraction and the real world. This alignment aids in the comprehension of the program by both you and your client. |

**REVIEW**

Market is a Java-based console application that simulates a basic market system. The application provides functionality to add users, show catalog, and make purchases.

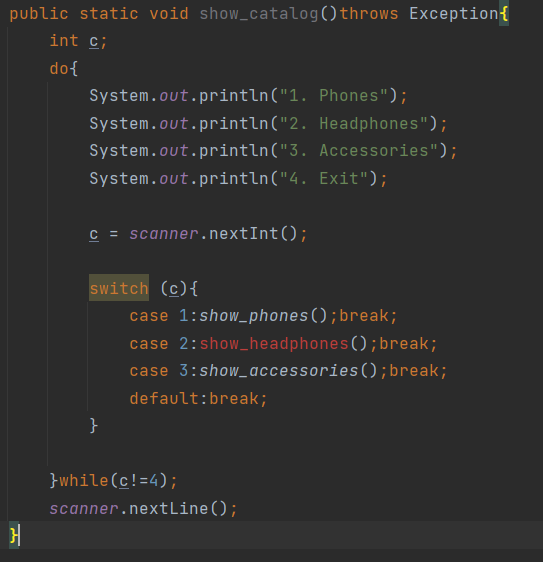
The application connects to a PostgreSQL database using JDBC, which provides the ability to interact with the database through SQL queries. The database has three tables, namely users, phones, and tablets, which store user information, phone information, and tablet information, respectively. The application also has two accessory tables, charger, and powerbank.

The application starts by presenting the user with a menu to add a user, show the catalog, or exit. If the user chooses to add a user, they are prompted to enter their name and balance. The application then inserts the user into the database and retrieves their id. The id is used throughout the application to identify the user.



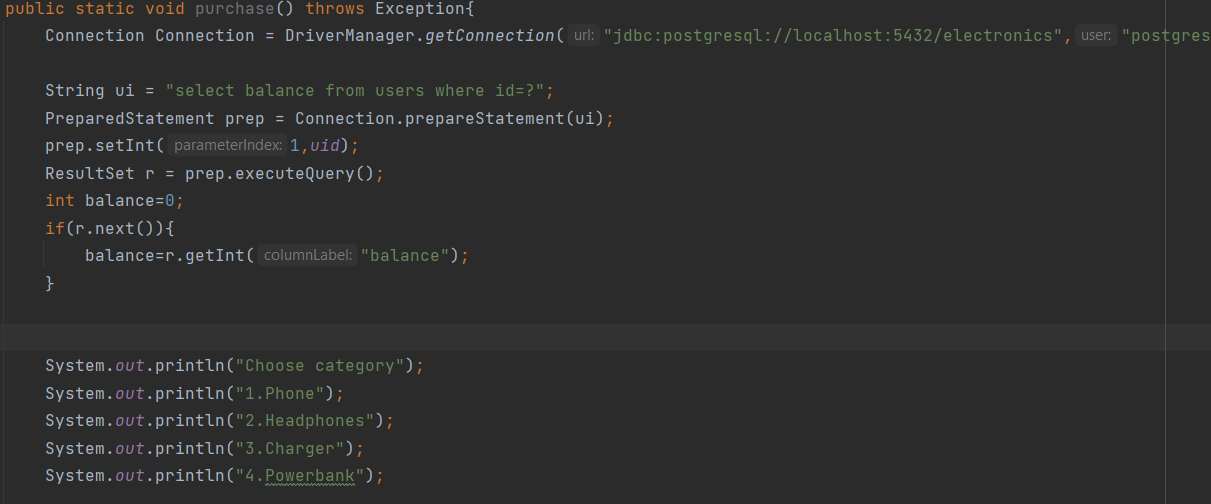
Picture 1.Main function

The user chooses to show the catalog, they are presented with a submenu to select the category they wish to view. The submenu includes options for phones, tablets, accessories, and an option to exit the submenu. If the user selects the phone option, they are presented with a submenu to select the brand they wish to view, namely Apple, Samsung, or Xiaomi. The application then retrieves the phones from the database for the selected brand and displays them to the user. If the user selects the tablet option, the application retrieves the tablets from the database and displays them to the user. If the user selects the accessory option, they are presented with a submenu to select the type of accessory they wish to view, namely charger or powerbank. The application then retrieves the accessories from the database for the selected type and displays them to the user.

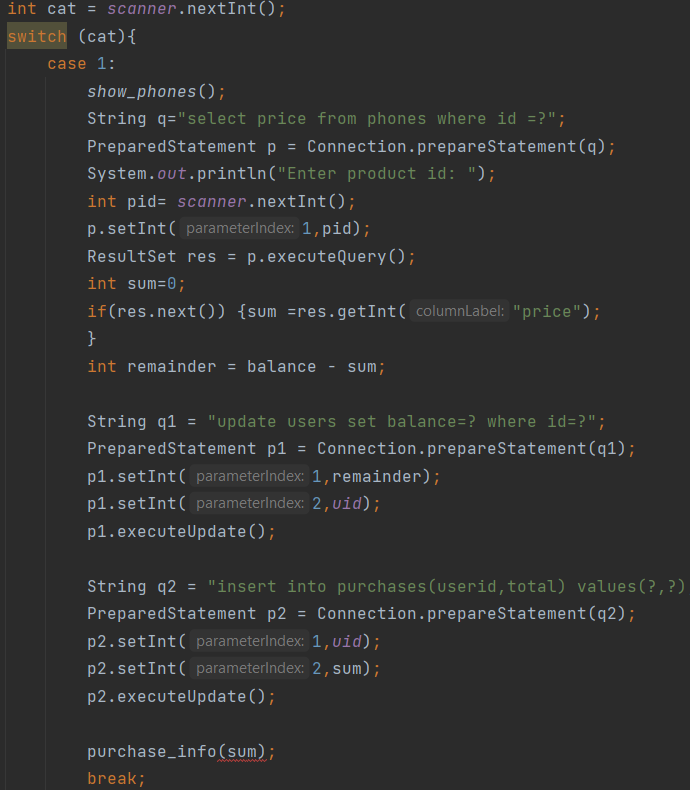


Picture 2. ”Show\_catalog” function

If the user chooses to make a purchase, they are presented with a submenu to select the category they wish to purchase, namely phone, tablet, charger, or powerbank. If the user selects the phone category, the application displays the available phones for the user to choose from. The user is prompted to enter the id of the phone they wish to purchase, and the application retrieves the price of the phone from the database. If the user's balance is sufficient, the application deducts the price of the phone from the user's balance and updates the balance in the database.

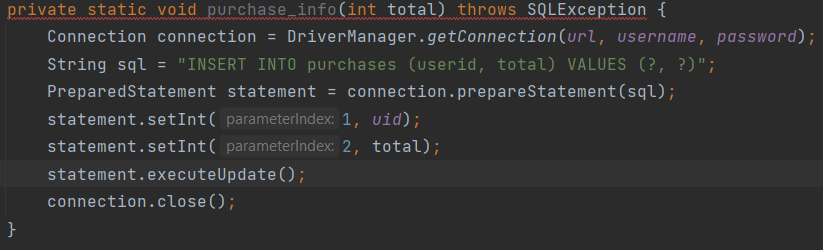


Picture 3. ”Purchase” function

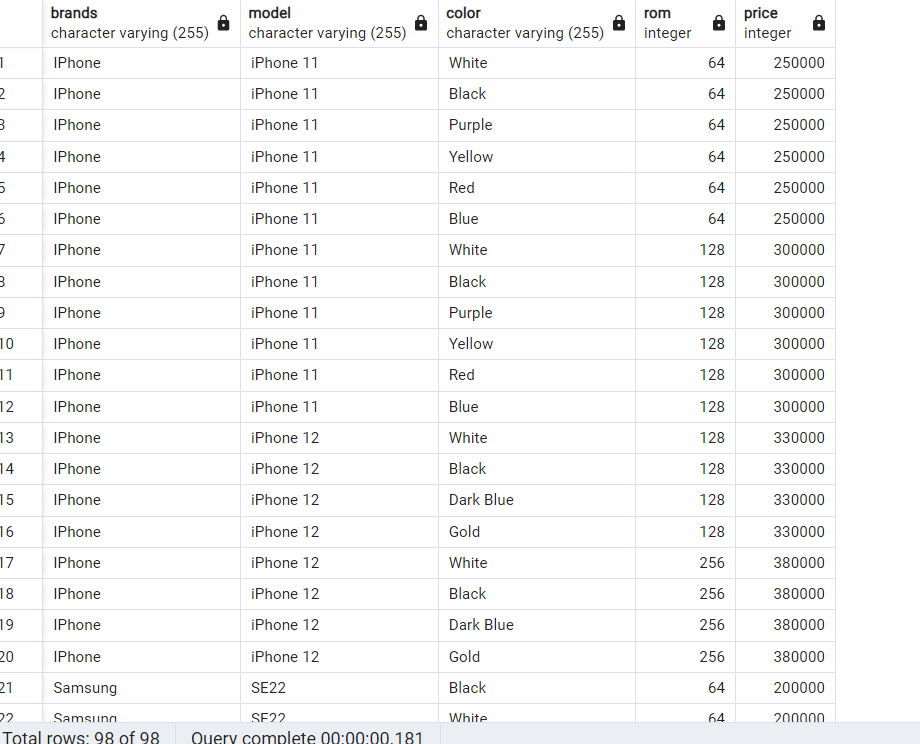
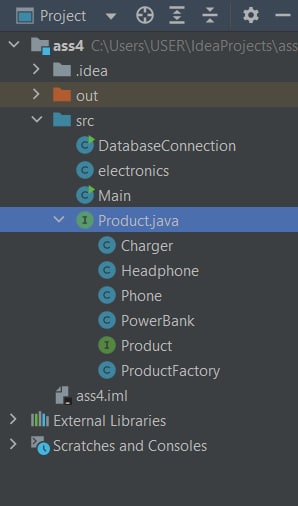


Picture 4. Case of “Purchase” function

The “purchase\_info” method is a private static method that takes an integer parameter called "total" and throws an SQLException. This method is responsible for inserting purchase information into the "purchases" table of a database.



Picture 5. ”purchase\_info” function

  
  
Picture 6. Table ”Phones”  
  
  


Picture 7. Structure of folder

PRODUCT FACTORY

A factory method is a generative design pattern that solves the problem of creating different products without specifying specific product classes.

The factory method specifies the method that should be used instead of calling the new operator to create product objects. Subclasses can override this method, to change the type of products created.

**CONCLUSION**

JDBC, a database connection, and S.O.L.I.D. were all employed in the develop ment of this program. OOP and programming patterns were two topics that were thoroughly studied. There have been studies on using code to interact with databases. There have been applied patterns.

**What did I enjoy about the project?**

Well, one of the things I appreciated was the opportunity to learn about version control in Git. Additionally, I found the project setup to be well-organized and effective. Through a combination of theoretical knowledge and practical implementation in each version, I gained a better understanding of design patterns. If it had been purely theoretical. Overall, the coursework was thoughtfully designed, and I felt better prepared. The problem-solution-consequences approach to learning about design patterns was especially helpful to me.

**What changes I would make if redesigning the same application?**

If I were to redesign the same application, there are several changes that I would consider making. Firstly, during a presentation, I was impressed by a classmate's approach of using server resource requests and request handling to design the system. Another change that I would make is separating the login and registration functionalities from the rest of the system, and having separate authorization and resource endpoints. Although I had previously considered splitting the model into two parts, i.e., authorization and functionality, using two RMI's, I was unable to do so due to time constraints and priorities.

In addition, I would avoid using RMI, as it has scalability issues and can only support a limited number of concurrent users. Instead, I would choose to use JSON or HashMap's for data exchange, as they are more practical choices for the future.

Lastly, if I had enough time to work on the assignment, I would implement synchronization without using the Java synchronized keyword in order to gain a deeper understanding of the design patterns. This change is motivated by my desire to learn more about the subject

**REFERENCE LIST**  
Java patterns  
<https://www.jmlr.org/papers/volume15/fournierviger14a/fournierviger14a.pdf>

Refactoring.Guru (n.d.). Паттерны/шаблоны проектирования.

https://refactoring.guru/ru/design-patterns