Hanoi University of Science and Technology

School of ICT

*

PROJECT REPORT

SUBJECT: OBJECT-ORIENTED PROGRAMMING

ELECTRONICS STORE MANAGEMENT

Student implementation: **Dinh Duc Lam 20198236**

Tran Duc Viet 20198275

Do Hoang Viet 20198272

Class number: **121723**

Instructor: **PhD. Do Thi Ngoc Diep**

Hanoi, May 2021

TABLE OF CONTENTS

ASSIGNMENTS

No	Name	Student id	Work assignment	Completion	Note
01	Đinh Đức Lâm	20198236	- Code and design *GUI* - Code package *function* - Write the report - Presentation	100%	None
02	Trần Đức Việt	20198275	- Code class *Cart, Customer, DeviceList, MainProcess* (in package *struct*)	100%	None
03	Đỗ Hoàng Việt	20198272	- Code class *Device, Laptop, Phone, History* (in package *struct*) - Design slide	100%	None

PROBLEM/SYSTEM INTRODUCTION

1. Problem introduction

An electronics store sells smartphone and laptop. Besides general information of a product such as name, brand, model, etc, there are also other infomations of smartphone are: **screen size, battery life and camera resolution**, other information of laptop are: **CPU, RAM, hard disk capacity**. When someone buy a production, the store will give a bill and save it into information system.

Building a store management system with the following functions:

- Add, remove, edit and delete a product in the store.
- Search for products by **name, brand**. Then displays all product information found.
- Payment function for customers to buy products, update products after buying/selling. Calculate the **sales/profit** of the store for an entered period input (based on the cost/sell price of products).

2. System introduction

A system is a store management has the functions given in the "Problem introduction" section. In addition, we added other following functions:

- Export excel file from laptop and smartphone data.

USED LIBRARY/PACKAGE PRESENTAION

1. Used library

1.1. Java util

Java util library provide tools for data handling such as adding, deleting, storing data,…

Java util are already available in the IDEs of Java, so it no need installing and easy to use.

1.2. Java AWT and Java Swing

Java AWT and Java Swing library provide GUI toolkit that is used to create *window-based applications* for a Java program.

Java AWT and Java Swing are already available in the IDEs of Java, so it no needs installing and easy to use.

1.3. Apache POI

Apache POI is an opensource library, is provided by Apache. It provides API to work with documents of Microsoft such as Word, Excel, PowerPoint, …

To use Apache POI library, we must download it from Apache's home page, then add it into classpath of Java. In Eclipse IDE, it will be installed by the following way:

- Right click on the project, choose *Properties.*
- In *Java Built Path* section, click on *Classpath*, then click on *Add JARs…*
- Select the path to Apache POI library, choose all the jar file in it, then click "OK".

2. Package presentation

Our program uses 3 packages with its own functions as follows:

2.1. Package struct

This package includes classes that represent the objects of the problem: *Device, Phone, Laptop, DeviceList, Customer, Cart, History:*

- 3 classes named *Device, Laptop (extends Device), Phone (extends Device)* to represent 2 objects (Laptop, Phone).
- Class *DeviceList* is a list of devices of either Laptop or Phone in store (the program uses 2 device list, one for Laptop, one for Phone) and performs all actions relate to device data: *add, edit, delete, search, export to Excel*.
- Class *Cart* is a list of Device was added to cart.
- Class *Customer* represent a customer that has 4 states: *Full Name, Address, Phone Number, a cart was bought by own.*
- Class *History* to saves all the transactions.

2.2. Package GUI

This package contains classes to build the GUI (Graphical User Interface) for the program: *Home, DeviceGUI, History, Cart, Bill, TextField, Button:*

- Class *Home* to create framework of the program.
- Class *DeviceGUI* to create where the user can interactive with Laptop and Phone data.
- Class *History* to create where the user can see history include sold device, sales, profit.
- Class *Cart* is a JFrame where user can fill in customer information who want to buy devices in the cart and show all devices in the cart.
- Class *Bill* show a bill after clicking on Pay button in *Cart* class.
- Classes named *TextField* and *Button* to define the JTextField and JButton to easy to use, there is no need to define repeating attributes of JTextField and JButton.

2.3. Package function

This package contains classes to develop other functions besides main functions of the program. In package *function,* we have just only a class *CreateExcel* to export Laptop data or Phone data to Excel file using Apache POI library.

2.4. Package icon

This package contains all icon of program.

SYSTEM ANALYSIS AND ACTIVITY DESCRIPTION

1. System analysis

The system has the full properties of an object-oriented system:

- Aggregation:
  - Each *DeviceList* class has a list of Device and *Cart* class is the same.
  - Each *Customer* class has a *Cart* class.
  - *History* class has a list of customers.
- Inheritance: Laptop and Phone are instances of Device.
- Abstraction: Class Device is an abstraction class.
- Polymorphism:
  - *getType()* and *getSTringArray()* methods in Device class were overridden in Laptop and Phone class.
  - *equals()* method of Objects class was overridden in Device class to differentiate devices from each other.
- Encapsulation:
  - Classes in the program were bundled of data with the methods (or other functions) operating on that data.
  - All states of class were declared private or protected.
- Upcasting and Downcasting:
  - See a *Device* object as a *Laptop* or *Phone* object.
  - Converting *Device* to *Laptop* or *Phone*.

The system has classes represent objects in reality:

- Devices
  - Has the states of Device.
- List of Devices
  - Has many Devices.
  - Add a Device into list.
  - Remove a Device from list.
  - Modify a Device in list.
  - Search Devices by some states.
- Customers (This is a store device management program, not a sales program. So, the customer is designed from the store's perspective, a customer in this program is only used to save information and easily manage, no actions such as payment, etc.)
  - Has a states of human (full name, address, phone number, etc).
  - Has a shopping cart that this customer bought.
- Shopping cart (almost like a small list if Devices)
  - Has Devices was added into shopping cart.
  - Total profit, total sales if this cart is sold.
  - Add a Device into shopping cart (Customer buy this Device).
- History
  - Has customers who bought in this store.

- Has Devices were bought by customers.

- Store
  - Has some lists of Device.
  - Transfer the Device to the cart for the customer.
  - Receive payment orders and add it in History.

2. Activity description

After starting the program, the program's interface will appear and allow the user to use the following functions:

(1) Laptop Button: When the button laptop is clicked, all the data of laptops can be used for work.

(2) Phone Button: Similar to the functions of the Laptop Button, the phones' data can be used for work.

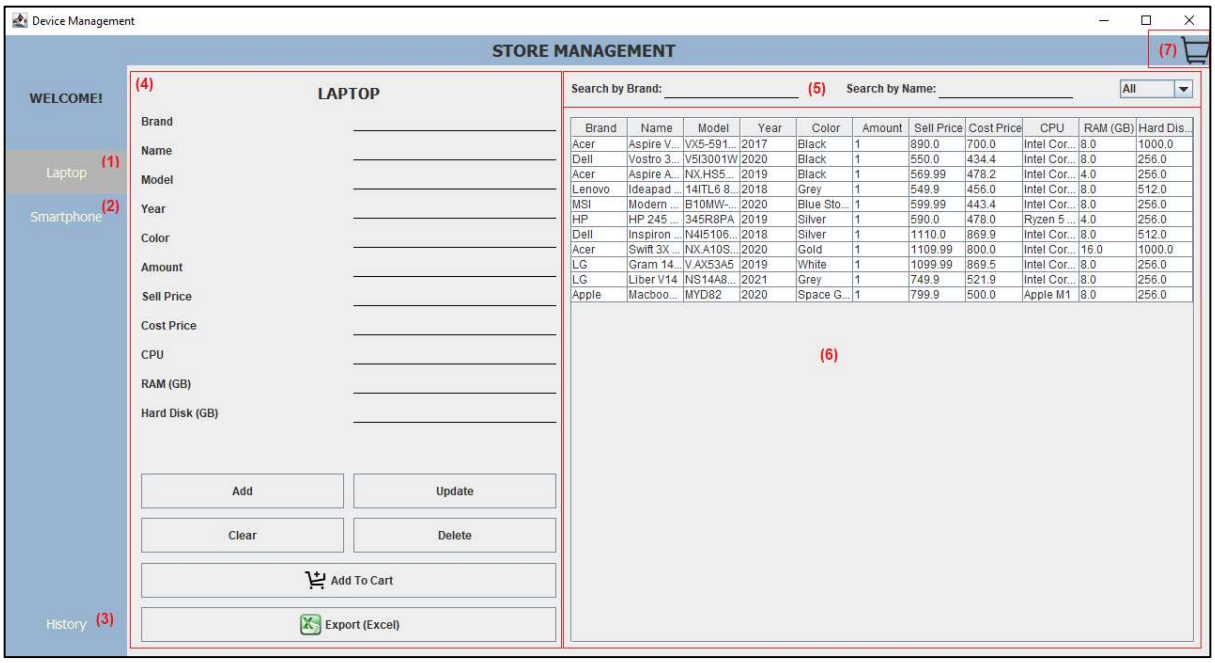
(3) History Button: if the history button is clicked, all transactions, sales and profit will be shown.

(4) There are functions to add, delete, edit and add to cart and function that can turn the product data into the excel file.

(5) There is a function to search for the product.

(6) There is a table show the information of all products.

(7) Is a button which clicks to view all Device are being in the cart.



The interface of the program after booting

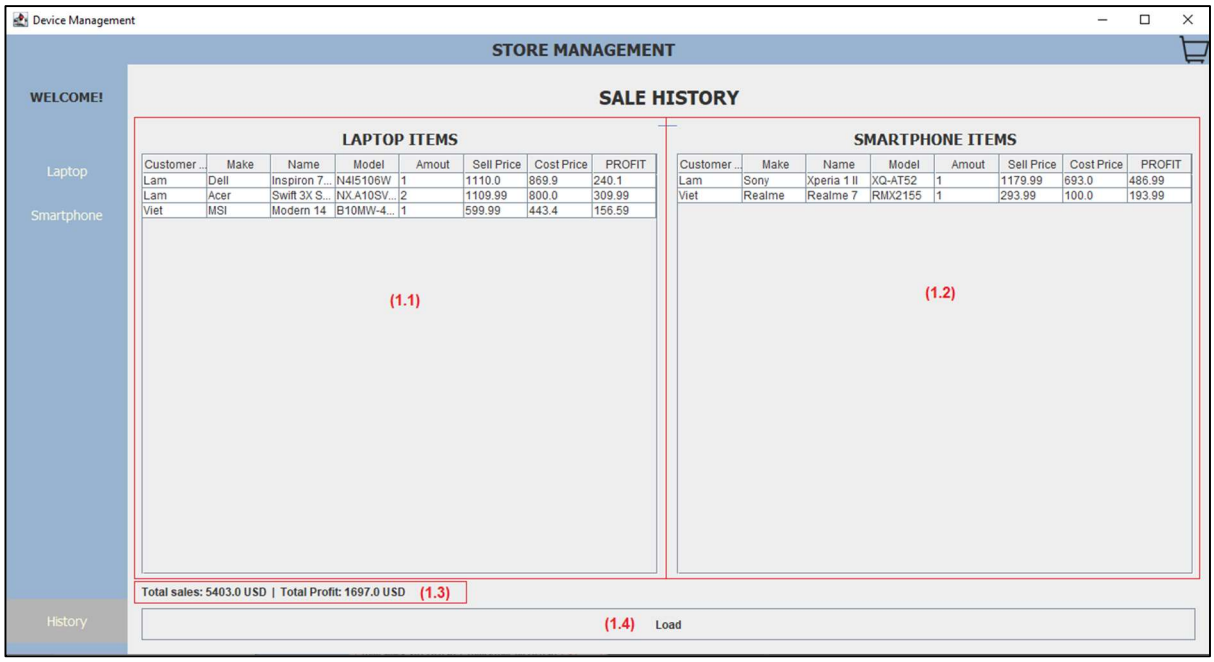
If the History button (3) is clicked, the history interface will be shown as the picture below:

(1.1) Sold laptop items.

(1.2) Sold phone items.

(1.3) Total sales and profit.

(1.4) Load button: when this button is clicked, the sale history will be updated.



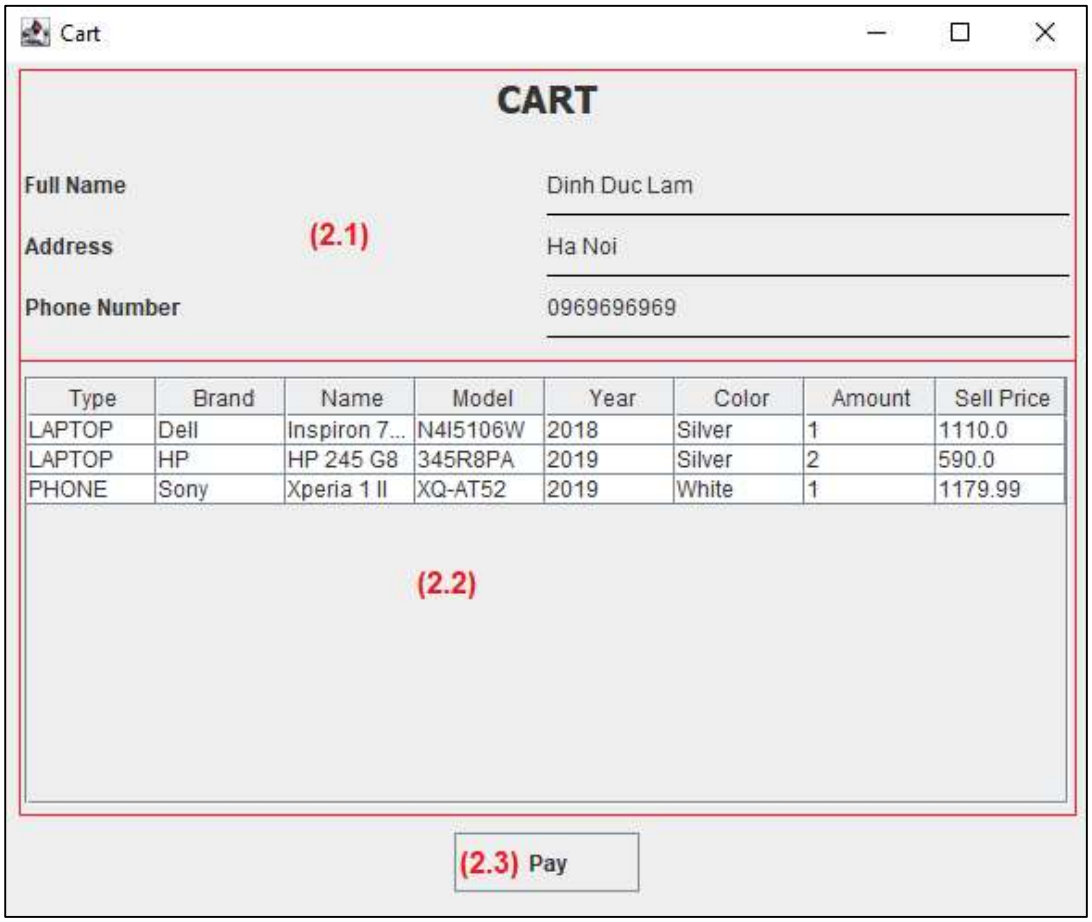
The history interface

If cart button (7) is clicked, the cart interface will be show as the picture below:

(2.1) Where user fills in customer information.

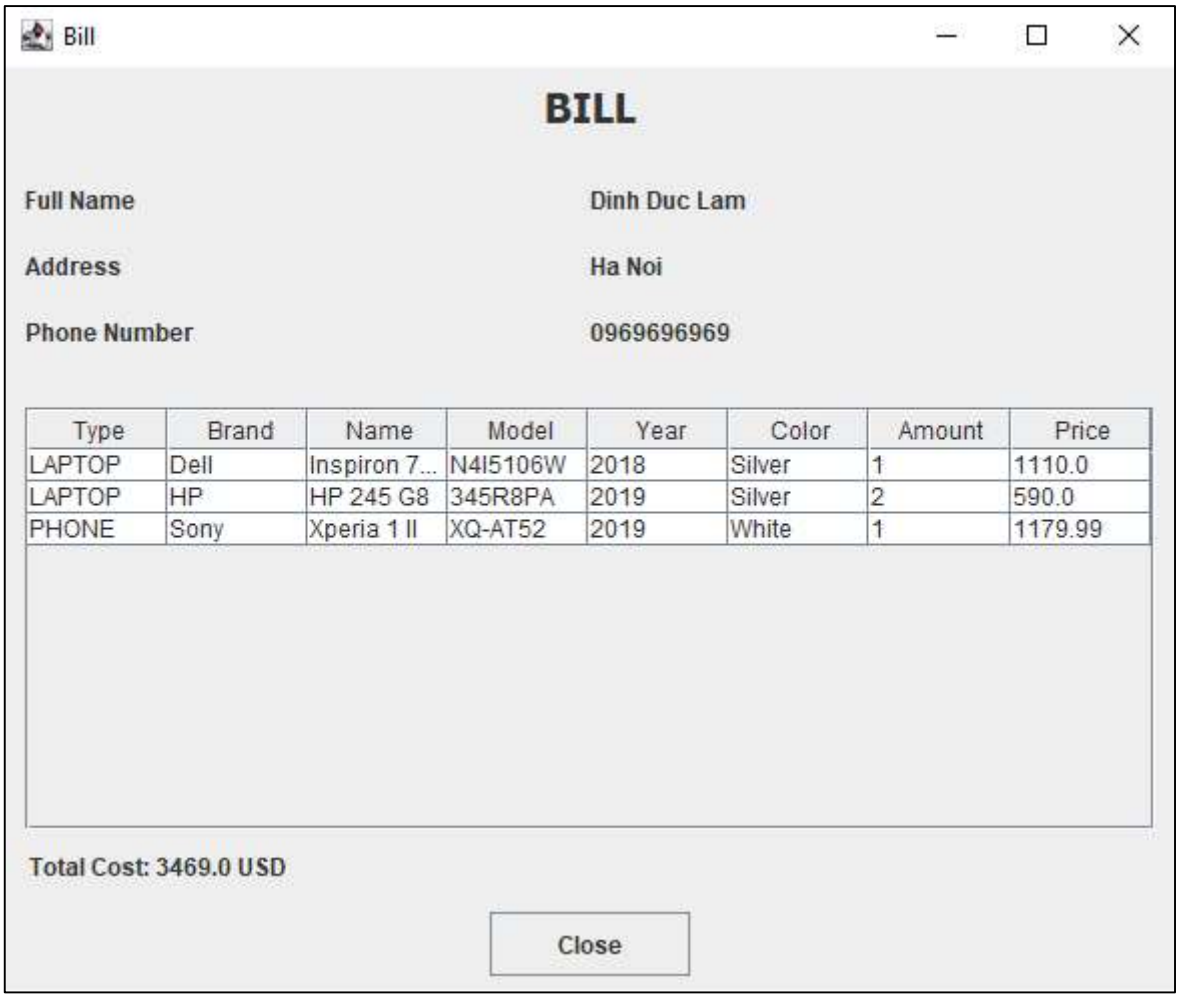
(2.2) All device that customer adds to cart.

(2.3) Button that clicks to pay for all device in the cart.



The cart interface

After filling in customer information, user click Pay to pay all devices in the cart. Then, a bill will appear. Or user can close the cart interface (all devices in the cart will be still intact).



The bill interface

CLASS DESCRIPTION

1. Class Device

- Class *Device* is an abstract class.
- Class *Device* contains the general states and general methods of Laptop and Smart Phone.
- LAPTOP_TYPE and PHONE_TYPE is an integer number representing the type of Device (Laptop is 0 and Phone is 1).
- LAPTOP_COLUMN_TITLE and PHONE_COLUMN_TITLE is a String array containing title of each column of the table data.
- Method *equals* override method equals of class Object to redefine the difference between two devices.

Device
brand : String # name : String # model : String # year : int # color : String # cellPrice : double # costPrice : double + LAPTOP_TYPE : int = 0 + PHONE_TYPE : int = 1 + LAPTOP_COLUMN_TITLE : String[] = { "Brand", "Name", "Model", "Year", "Color", "Amount", "Sell Price", "Cost Price", "CPU", "RAM (GB)", "Hard Disk (GB)" } + PHONE_COLUMN_TITLE : String[] = { "Brand", "Name", "Model", "Year", "Color", "Amount", "Sell Price", "Cost Price", "Screen Size (inchs)", "Battery Life (hous)", "Camera (MP)" }
+ Device(dv : Device) + Device(t : String[]) + *getType() : int* + *getStringArray() : String[]* + equals(Object obj) : boolean

2. Class Laptop and class Phone

- Classes *Laptop* and *Phone* are inherited from class *Device*.
- Method *getType()* return type of Device, so it helps us know if the device is a laptop or a phone.
- Method *getStringArray()* return a String array which each element of it is an attribute of that Device converted to String. For example, the method *getStringArray()* of class Laptop will return a String array of the form: { brand, name, model, year, color, cellPrice, costPrice, cpu, ram, disk } all elements of it was converted to String.
- Method *getStringArray()* helps us easy to handle Device data.

Laptop
cpu : String # ram : double # disk : double
+ getType() : int + getStringArray() : String[]

Phone
screen : double # battery : double # camera : double
+ getType() : int + getStringArray() : String[]

3. Class Customer

- Each class *Customer* has some states of a human: full name, address, phone number, etc.
- In addition, class *Customer* also has a shopping cart.

Customer
fullName : String # address : String # phoneNumber : String # cart : Cart
+ setInfo (fullName : String, address : String, phoneNumber : String) : void + getCart() : Cart + getFullName() : String + getAddress() : String + getPhoneNumber() : String

4. Class Cart

- Each class *Cart* like a small list of devices.
- Each class *Cart* has a totalCost and a totalProfit variables are calculated based on devices are in the cart.

Cart
list : List<Device> # totalCost : double # totalProfit : double
+ add(newDv : Device) : void + getList() : List <Device> + getTotalCost() : double

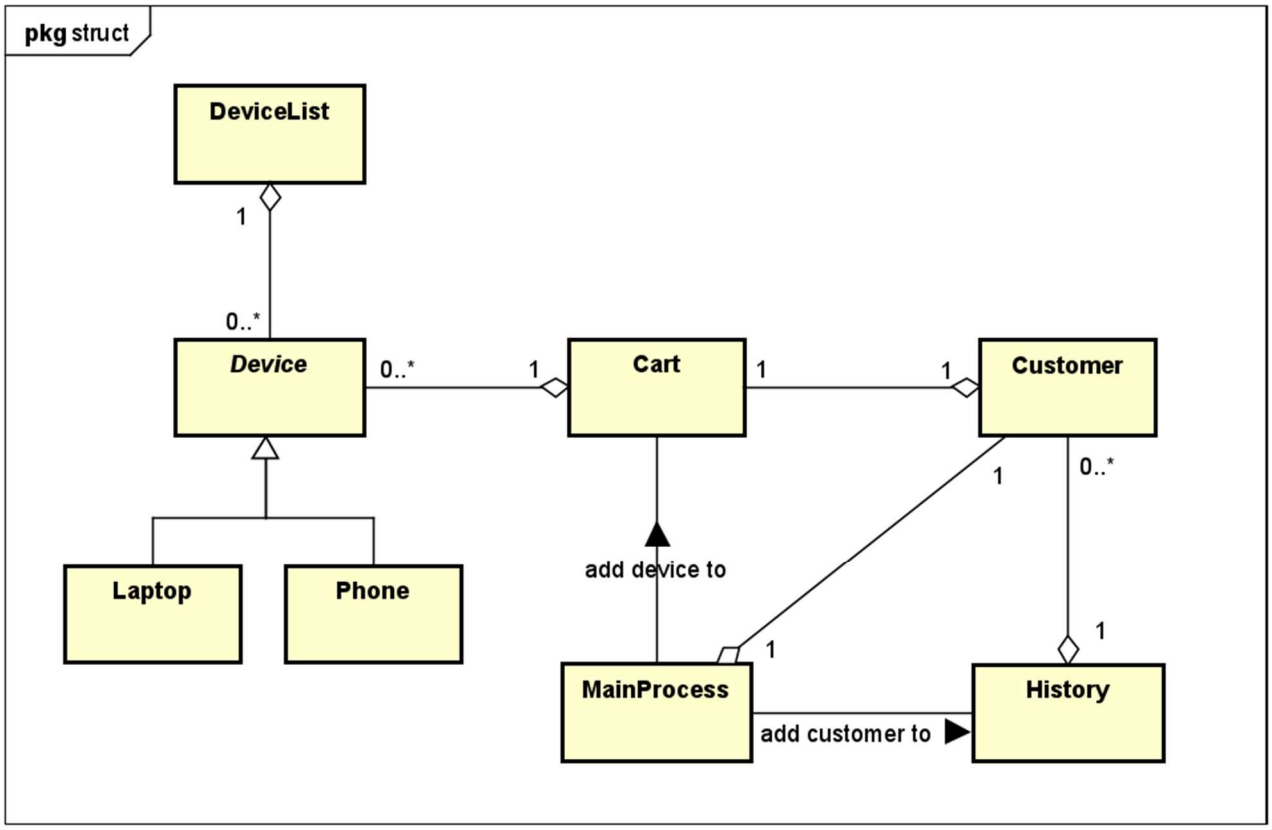
5. Class History

- A class *History* has a list of customers bought in this store.
- Each customer has a cart, so the system can show a list of sold items by getting items in the cart of each customer.
- All attributes and methods in class *Customer* are abstraction.

History
customers : List<Customer> # sales : double # profit : double
+ add(customer : Customer) : void + getCustomers() : List <Customer> + getProfit() : double + getSales() : double

SYSTEM/CLASS DESIGN

1. Class diagram



Class diagram of package *struct*

SELF-ASSESSMENT OF RESULTS, LIMITATIONS

1. Self-assessment of results

- Completed all the requests have been set.
- Have aggregation, inheritance and polymorphism properties in system.
- Construction all classes in the system, with complete attributes and methods.
- Built the corresponding polymorphism methods.
- Completed the Class diagram in UML.
- Completed the system with GUI.

2. Limitations

- Have not developed the nesscessary functions yet.
- The graphics are not good enough.