

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



ĐỒ ÁN CUỐI KỲ BỘ MÔN MẪU THIẾT KẾ

HỆ THỐNG QUẢN LÝ SHOP QUẦN

ÁO

Người hướng dẫn: **ThS. NGUYỄN THANH PHƯỚC**

Người thực hiện: **PHAN AN DUY – 518H0616**

LÊ TẤN TÀI – 518H0114

Lớp : 18H50301 – 18H50302

Khoá : 22

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

**TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN**



ĐỒ ÁN CUỐI KỲ BỘ MÔN MẪU THIẾT KẾ

HỆ THỐNG QUẢN LÝ SHOP QUẦN

ÁO

Người hướng dẫn: **ThS. NGUYỄN THANH PHƯỚC**

Người thực hiện: **PHAN AN DUY**

LÊ TẤN TÀI

Lớp : **18H50301 – 18H50302**

Khoá : **22**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021

PHẦN ĐÁNH GIÁ CỦA GIẢNG VIÊN

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

LỜI CẢM ƠN

Để hoàn thành quá trình nghiên cứu và hoàn thiện bài báo cáo này, em xin gửi lời cảm ơn chân thành và sâu sắc đến ThS Nguyễn Thanh Phước - giảng viên môn Mẫu Thiết Kế, trường Đại học Tôn Đức Thắng. Thầy đã trực tiếp chỉ bảo và hướng dẫn em trong suốt quá trình nghiên cứu để em hoàn thiện bài báo cáo này.

Trân trọng cảm ơn!

MỤC LỤC

PHẦN ĐÁNH GIÁ CỦA GIẢNG VIÊN	i
LỜI CẢM ƠN	1
MỤC LỤC	2
PHẦN 1: GIỚI THIỆU TỔNG QUAN CHƯƠNG TRÌNH	3
PHẦN 2: CÁC MẪU THIẾT KẾ ĐƯỢC ÁP DỤNG	4
2.1 Mẫu thiết kế Singleton	4
2.2 Mẫu thiết kế Template Method	5
2.3 Mẫu thiết kế Factory Method	6
2.4 Mẫu thiết kế Observer	7
2.5 Mẫu thiết kế MVC	8
PHẦN 3: ĐOẠN CODE ĐƯỢC THỰC THI	10
3.1 Code Singleton Pattern	10
3.2 Code Factory Method Pattern	10
3.3 Code Template Method Pattern	11
3.4 Code Observer	13
3.5 Code MVC Pattern	14
Link project được đăng lên github:	15
PHẦN 4: DEMO	17
TÀI LIỆU THAM KHẢO	20

PHẦN 1: GIỚI THIỆU TỔNG QUAN CHƯƠNG TRÌNH

Đề tài chọn cho bài tập lớn là Quản lý cửa hàng bán quần áo. Với nhu cầu mua quần áo nhiều hơn từ người dùng, từ ban đầu chúng ta có thể quản lý được tất cả những mặt hàng quần áo, từ số lượng cho đến vị trí chỉ bằng cây viết và những tờ giấy. Nhưng khi càng nhiều khách hàng muốn mua quần áo nhiều hơn, cung và cầu càng ngày càng nhiều hơn. Nên quản lý một lượng đồ rất lớn có thể gây rối cho người quản lý. Nếu sử dụng cách truyền thống, cửa hàng sẽ gặp khó khăn trong việc kiểm kê đơn hàng trong ngày hôm đó, và số lượng đơn hàng mà các khách hàng đã đặt nên chúng em đã lên ý tưởng cho 1 phần mềm để quản lý các dữ liệu đơn hàng.

Các chức năng sau đây sẽ giúp ích rất nhiều cho một cửa hàng

- Cho phép cửa hàng có thể đăng nhập và đăng xuất
- Tạo các hóa đơn cho khách hàng, thêm và bớt được các món hàng
- Liệt kê ra được thành danh sách hoặc thành bảng
- Cho phép kiểm tra lại nếu khách hàng có yêu cầu.
- Hóa đơn tạo bị lỗi hoặc hóa đơn nháp có thể được xóa bỏ
- Hóa đơn phải hỗ trợ các loại thanh toán khác nhau như tiền mặt, thẻ và loại hình hóa đơn thu hộ (COD)
- Cho phép chủ cửa hàng thêm các mặt hàng mới vào kho, từ đó có thể thêm vào hóa đơn khi bán cho khách.
- Cho phép đăng xuất khỏi phần mềm khi chủ cửa hàng không hiện diện trước máy tính.

PHẦN 2: CÁC MẪU THIẾT KẾ ĐƯỢC ÁP DỤNG

Một chương trình chúng ta có thể đơn giản viết thành một file Main bao gồm tất cả các functions, các logic toán học, hay những yêu cầu của requirement mà người ra đề yêu cầu làm. Nhưng điểm trừ khi làm việc đó là chương trình sẽ trở nên rất là rối, dài và khó kiểm soát. Vì thế, những người trước đã cho ra những mẫu thiết kế, để tăng mức độ flexibility cho một chương trình, đơn giản hóa và khai thác toàn bộ chức năng của lập trình hướng đối tượng (Object-Oriented Programming)

Trong yêu cầu bài của chúng em, thì phần mềm quản lý shop quần áo cũng được thiết kế chương trình theo các mẫu design pattern đang có trên cộng đồng. Với ngôn ngữ chọn là Java, thì chúng em đã thiết kế chương trình này với những mẫu Pattern sau:

- **Singleton Pattern**
- **Factory Pattern**
- **Observer Pattern**
- **Model-View-Controller (MVC)**
- **Strategy Pattern**

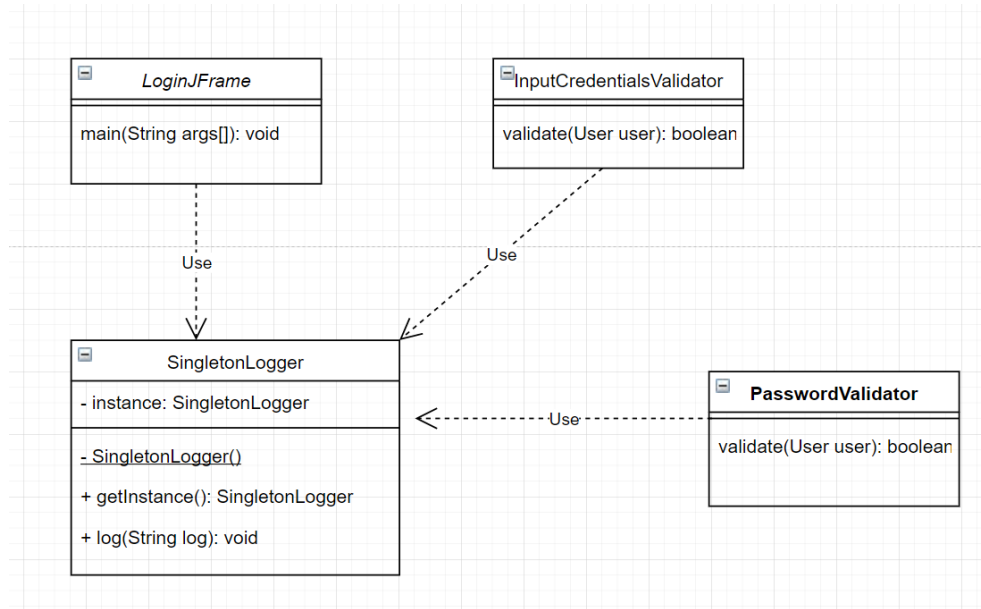
Chúng em sẽ trình bày cụ thể hơn về các pattern được liệt kê ở trên.

2.1 Mẫu thiết kế Singleton

Mẫu thiết kế Singleton là mẫu thiết kế đơn giản nhất trong các loại mẫu thiết kế, thuộc nhóm Creational Pattern. Nó cho phép chúng ta viết 1 instance trong 1 class và trả về instance đó mà các class khác khai về. Với mục đích làm đơn giản hóa function khi chúng ta khởi tạo và tránh việc chúng ta phải khai báo lại 1 lần nữa.

Với Singleton Pattern, chúng em đã sử dụng trong việc thiết kế phần login cho chương trình. Lý do chọn Singleton Pattern cho phần vì nó đơn giản, phù hợp cho việc khai báo check dữ liệu và tránh việc sử dụng dư bộ nhớ.

Sơ đồ thể hiện mẫu thiết kế Singleton:



Hình 2.1: Sơ đồ class áp dụng mẫu thiết kế Singleton

Qua sơ đồ ta có thể thấy một instance được khởi tạo và được gán private – để tránh trường hợp khai báo đề lên tính năng và làm tràn bộ nhớ.

Kế đó, khai báo hàm SingletonLogger() để sử dụng instance vừa được lập và trả về Logger sử dụng getInstance.

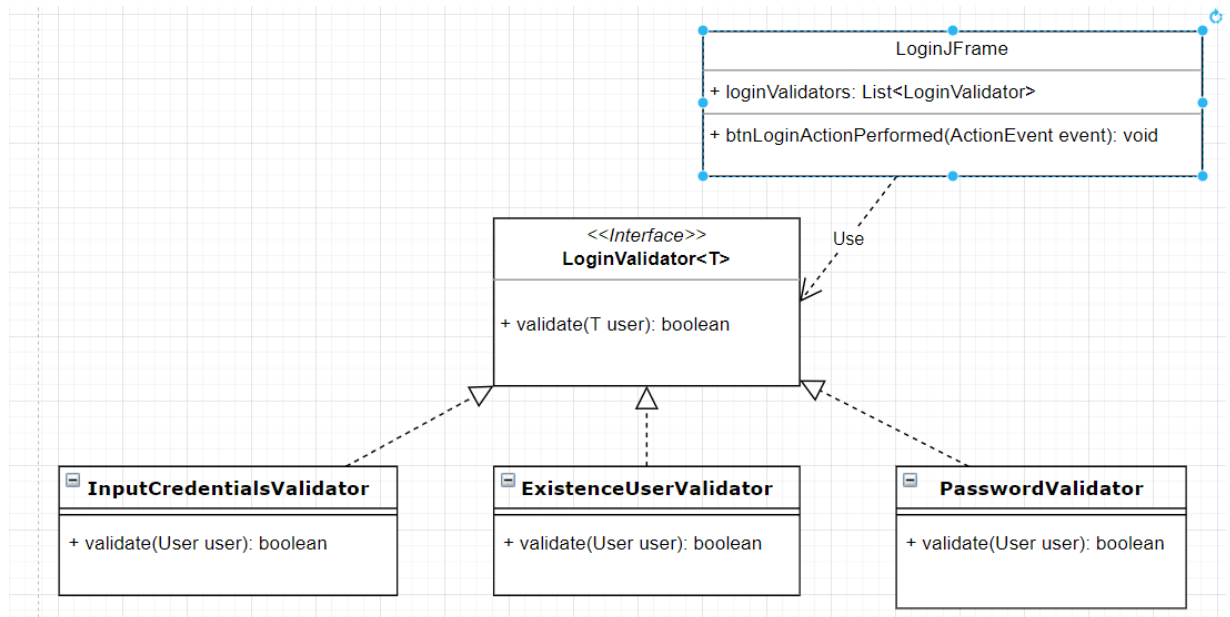
SingletonLogger là một lớp tiện ích bao bọc chức năng in thông báo ra màn hình, bên trong Phương thức SingletonLogger#log(String message) sử dụng System.out.println();

SingletonLogger được sử dụng nhiều nơi trong chương trình để in các thông báo, việc khởi tạo nhiều thực thể của lớp này là không cần thiết và lãng phí bộ nhớ. Ngoài ra, cho mục tiêu mở rộng sau này, SingletonLogger#log(String message) có thể dùng để ghi logs vào các tập tin và các bên thứ 3 như Spunk, ElasticSearch..., việc giữ cho 1 thực thể duy nhất tồn tại trong lúc chương trình hoạt động cũng đảm bảo tính toàn vẹn của dữ liệu được truyền đi. Do đó, việc áp dụng mẫu Singleton vào việc ghi logs là cần thiết.

2.2 Mẫu thiết kế Template Method

Template có thể được xem là một dạng mẫu thiết kế thuộc nhóm hành vi (Behavioral Pattern)

Khi người dùng nhập vào username và password, chương trình cần thực hiện một số dạng kiểm tra như chuỗi nhập vào có thỏa các ràng buộc về độ dài, ký tự cho phép, tên username có tương ứng với 1 đối tượng User tồn tại trong hệ thống và mật khẩu có đúng hay không. Qui trình các loại kiểm tra đều diễn ra giống nhau về mặt khung sườn: nhận dữ liệu là 2 chuỗi, kiểm tra, trả về kết quả đúng hoặc sai (true/false). Để tránh việc mã code lập đi lập lại khung sườn (duplicate code), mẫu Template method có thể được áp dụng. Ngoài ra từng lớp cụ thể (InputCredentialsValidator, ExistenceUserValidator, PasswordValidator) còn giúp mã chương trình trở nên minh bạch, rõ ràng lớp nào xử lý phần công việc nào.

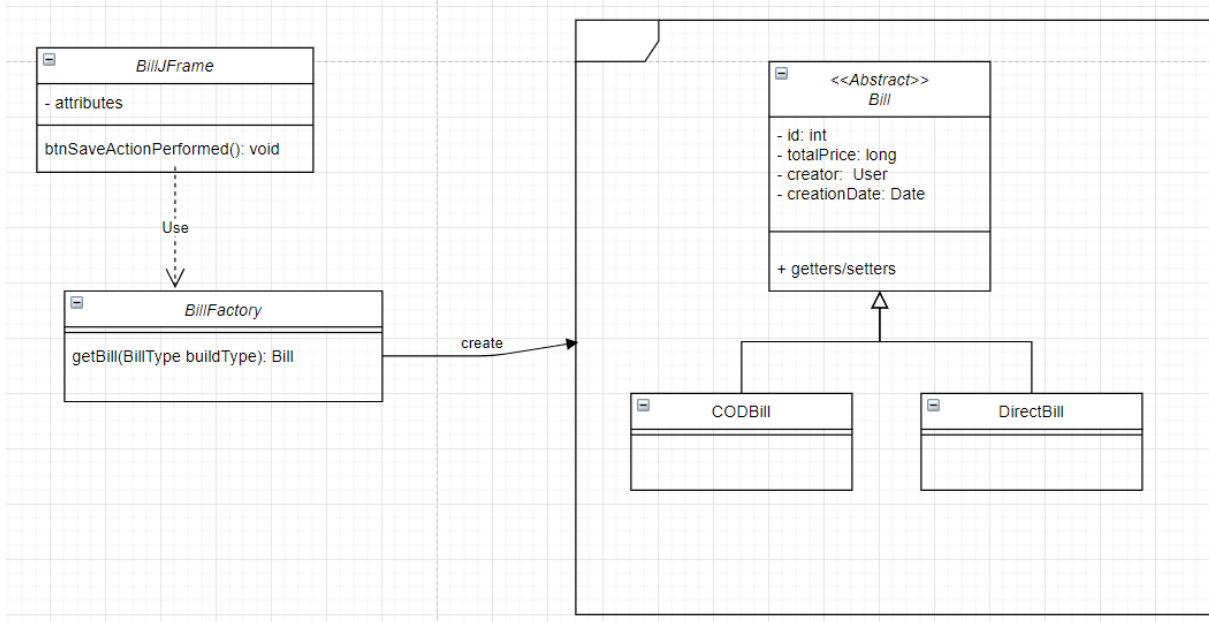


Hình 2.2: Sơ đồ lớp áp dụng Template Method Pattern

2.3 Mẫu thiết kế Factory Method

Việc kinh doanh trong cửa hàng có thể sẽ phát sinh nhiều loại hóa đơn khác nhau, hóa đơn sỉ/lẻ, trả trực tiếp hoặc hóa đơn thu hộ... Các đối tượng hóa đơn này sẽ được khởi tạo và sử dụng ở nhiều nơi khác nhau trong chương trình (thêm/sửa/xóa). Khó khăn sẽ xảy

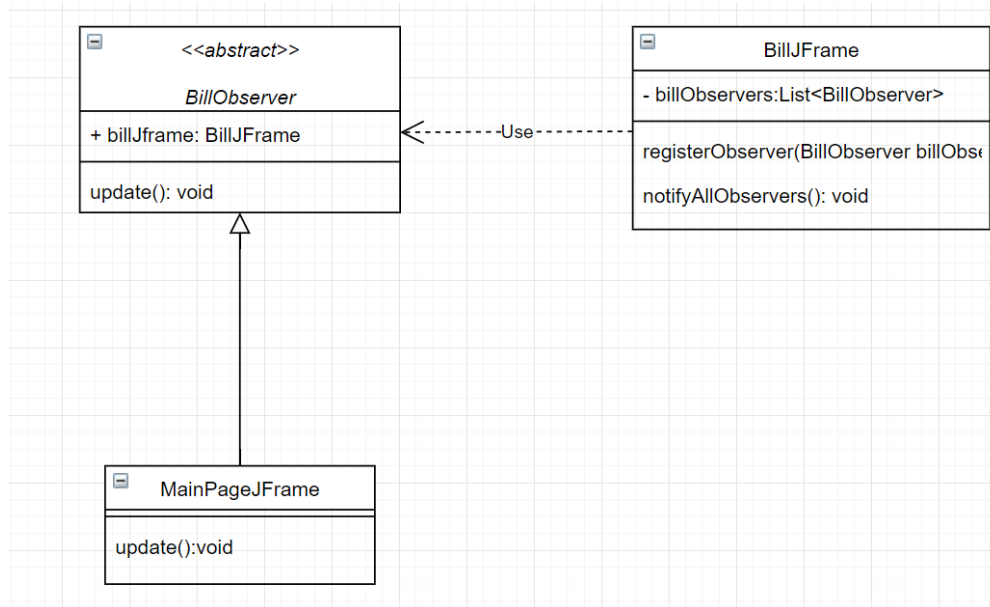
ra khi việc kinh doanh phát sinh loại hóa đơn mới hoặc thay đổi cách thức khởi tạo 1 hóa đơn. Để thuận lợi cho việc bảo trì và mở rộng, việc khởi tạo các loại hóa đơn nên áp dụng Factory Method pattern, phần code cho việc khởi tạo các đối tượng này được tập trung tại 1 chỗ duy nhất trong lớp BillFactory.



Hình 2.3 Sơ đồ lớp áp dụng Factory Method Pattern

2.4 Mẫu thiết kế Observer

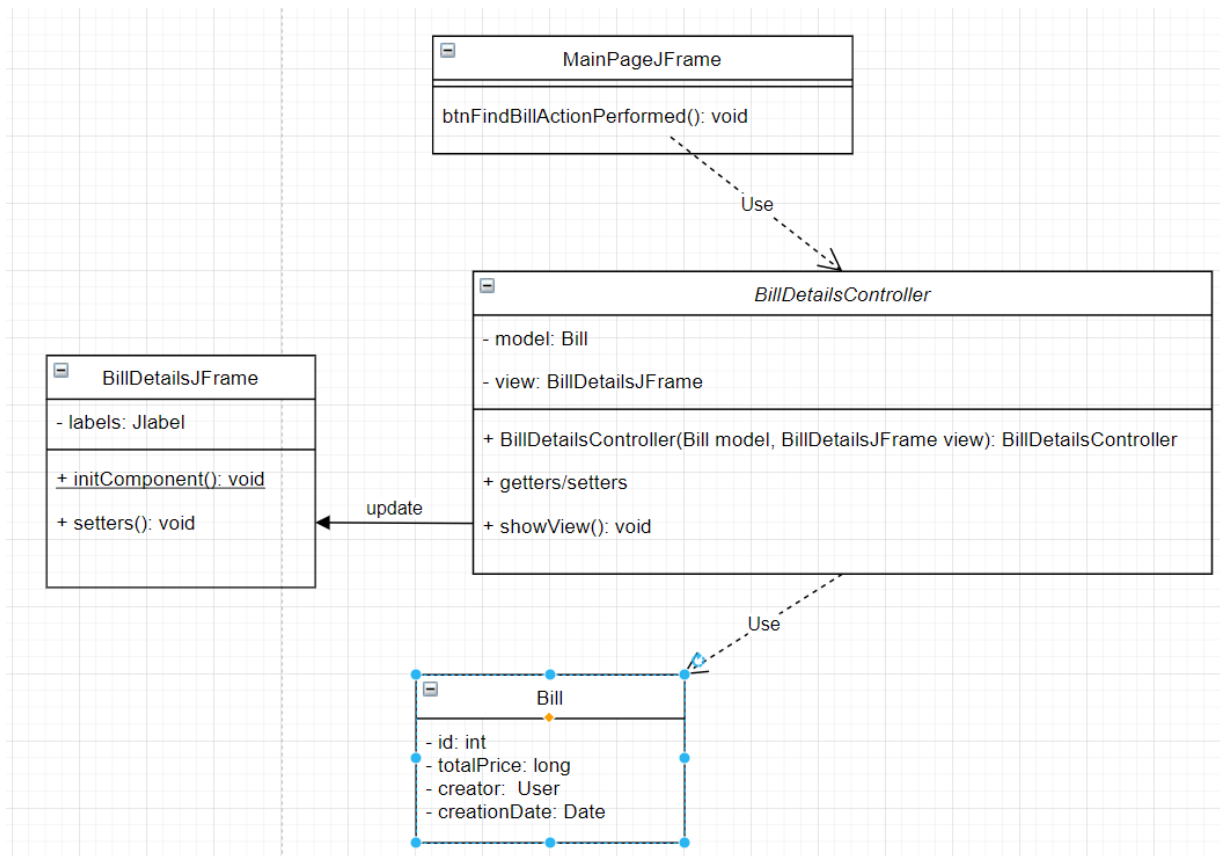
Chương trình cho hiển thị các thực thể của các hóa đơn trên 1 bảng trong trang chủ. Khi danh sách các thực thể thay đổi do thêm/xóa ở một trang giao diện khác (BillJFrame), bảng trên trang chủ cần phải được cập nhật tương ứng, để đáp ứng trường hợp này, Observer là mẫu phù hợp, bảng trên trang chủ (MainPageJFrame) là thực thể cần theo dõi (observe/watch) danh sách các hóa đơn trên các trang thêm/xóa. Như vậy, mỗi khi danh sách các hóa đơn thay đổi, trang chủ sẽ được thông báo và cập nhật tương ứng.



Hình 2.4: Sơ đồ lớp áp dụng Observer Pattern

2.5 Mẫu thiết kế MVC

Một thực thể Bill được hiển thị các giá trị các thuộc tính trên trang BillDetailsJFrame trong tính năng tìm kiếm, Nếu thực thể (Bill) được xử lý cụ thể ngay trong lớp giao diện (BillDetailsJFrame) chương trình vẫn đáp ứng yêu cầu hiển thị, tuy nhiên, sẽ gây khó khăn cho việc mở rộng và thay đổi giao diện hoặc thuộc tính các thực thể do tính gắn kết cứng giữa giao diện (BillDetailsJFrame) và dữ liệu của thực thể (Bill) trong cùng một tập tin java (BillDetailsJFrame). MVC có thể dung trong trường hợp này để giảm đi tính gắn kết chặt giữa 2 thành phần trên bằng cách tạo ra 1 lớp trung gian (BillDetailsController) để đảm nhận việc đọc dữ liệu từ Bill và cập nhật vào BillDetailsJFrame. Như vậy việc thay đổi lớp giao diện hoặc thực thể cũng trở nên dễ dàng và rõ ràng hơn.



Hình 2.5: Sơ đồ class thể hiện mẫu thiết kế MVC Pattern

PHẦN 3: ĐOẠN CODE ĐƯỢC THỰC THI

Phần 3 được dành riêng để thể hiện code được thực thi. Những đoạn code sau là những đoạn code áp dụng các mẫu thiết kế được liệt kê ở trên

3.1 Code Singleton Pattern

```

1 package dp.tdtu.singletonlogger;|
2
3 public class SingletonLogger {
4
5     //create an object of SingleObject
6     private static SingletonLogger instance = new SingletonLogger();
7
8     //make the constructor private so that this class cannot be
9     //instantiated
10    private SingletonLogger(){}
11
12    //Get the only object available
13    public static SingletonLogger getInstance(){
14        return instance;
15    }
16
17    public void log(String message){
18        System.out.println(message);
19    }
20 }

```

Hình 3.1: Code Singleton với hàm SingletonLogger()

3.2 Code Factory Method Pattern

```

1 package dp.tdtu.factory;
2
3 import dp.tdtu.model.Bill;
4
5
6
7
8 public class BillFactory {
9
10     public Bill getBill(BillType buildType){
11
12         if(BillType.DIRECT.equals(buildType)){
13             return new DirectBill();
14
15         } else if (BillType.COD.equals(buildType)){
16
17             Bill codBill = new CODBill();
18             codBill.setIsPaid(false);
19             codBill.setPaidDate(null);
20             return codBill;
21         }
22
23         return null;
24     }
25 }
26

```

Hình 3.2: Code Factory với class BillFactory()

3.3 Code Template Method Pattern

```

1
2 package dp.tdtu.validator;
3
4 public interface LoginValidator<T> {
5     boolean validate(T user);
6 }
7

```

Hình 3.3.1: Khởi tạo interface LoginValidator

```

1 package dp.tdtu.validator;
2
3 import dp.tdtu.model.User;
4
5
6
7 public class ExistenceUserValidator implements LoginValidator<User> {
8
9     UserRepository userRepo = new UserRepository();
10
11     @Override
12     public boolean validate(User user) {
13         User foundUser = userRepo.findUserByUsername(user.getUsername());
14         if(foundUser == null){
15             SingletonLogger.getInstance().log("User does not exist " + user.getUsername());
16             return false;
17         }
18         return true;
19     }
20
21 }

```

Hình 3.3.2: Code kiểm tra Username

```

1 package dp.tdtu.validator;
2
3 import dp.tdtu.model.User;
4
5
6 public class InputCredentialsValidator implements LoginValidator<User> {
7
8     @Override
9     public boolean validate(User user) {
10         if(user.getUsername() == null
11             || user.getPassword() == null
12             || user.getUsername().isEmpty()
13             || user.getPassword().isEmpty()
14             || user.getUsername().contains(" ")
15             || user.getPassword().contains(" ")){
16             SingletonLogger.getInstance().log("The user name and password are not following the restrictions");
17             return false;
18         }
19         return true;
20     }
21 }

```

Hình 3.3.3: Code kiểm tra Textbox đăng nhập

```

1
2 package dp.tdtu.validator;
3
4 import dp.tdtu.model.User;
5
6
7
8 public class PasswordValidator implements LoginValidator<User> {
9     UserRepository userRepo = new UserRepository();
10
11     @Override
12     public boolean validate(User user) {
13         User foundUser = userRepo.findUserByUsername(user.getUsername());
14         if(!foundUser.getPassword().equals(user.getPassword())){
15             SingletonLogger.getInstance().log("Password is incorrect");
16             return false;
17         }
18         return true;
19     }
20
21 }

```

Hình 3.3.4: Code kiểm tra Password

3.4 Code Observer

```

1
2 package dp.tdtu.ui;
3
4 import javax.swing.JFrame;
5
6
7 public abstract class BillObserver extends JFrame{
8     protected BillJFrame billJframe;
9     protected abstract void update();
10 }
11

```



```

12 8+ import dp.tdtu.model.Bill;
13
14 14- /**
15 15  *
16 16  * @author Netbean <2021>
17 17  */
18 18 public class MainPageJFrame extends BillObserver {
19
20 20-     /**
21 21  * Creates new form MainPageJFrame
22 22  */
23 23 private User user;
24 24- public MainPageJFrame(User user) {
25 25     initComponents();
26 26     this.user = user;
27 27     loggedInUserName.setText(user.getUsername());
28 28     loggedInFullname.setText(user.getFullName());
29 29
30 30     initTable();
31 31 }
32

```

Hình 3.4 Code áp dụng mẫu Observer

3.5 Code MVC Pattern

```

6 package dp.tdtu.ui;
7
8 import dp.tdtu.model.Bill;
9
10 /**
11  *
12  * @author Netbean <2021>
13  */
14 public class BillDetailsController {
15
16     private Bill model;
17     private BillDetailsJFrame view;
18
19     public BillDetailsController(Bill bill, BillDetailsJFrame billDetailsView) {
20         this.model = bill;
21         this.view = billDetailsView;
22     }
23
24     public Bill getModel() {
25         return model;
26     }
27
28     public void setModel(Bill model) {
29         this.model = model;
30     }
31
32     public BillDetailsJFrame getView() {
33         return view;
34     }
35
36     public void setView(BillDetailsJFrame view) {
37         this.view = view;
38     }
39
40     public void showView(){
41         view.setLblBillCreator(model.getCreator().getUsername() + " - " + model.getCreator().getFullName());
42         view.setLblBillDate(model.getCreationDate().toString());
43         view.setLblBillId(String.valueOf(model.getId()));
44         view.setLblBillPaid(model.isIsPaid() ? "Paid" : "Unpaid");
45         view.setLblBillTotalPrice(String.valueOf(model.getTotalPrice()));
46         view.setVisible(true);
47     }
48 }

```

Hình 3.5 Code áp dụng mẫu MVC

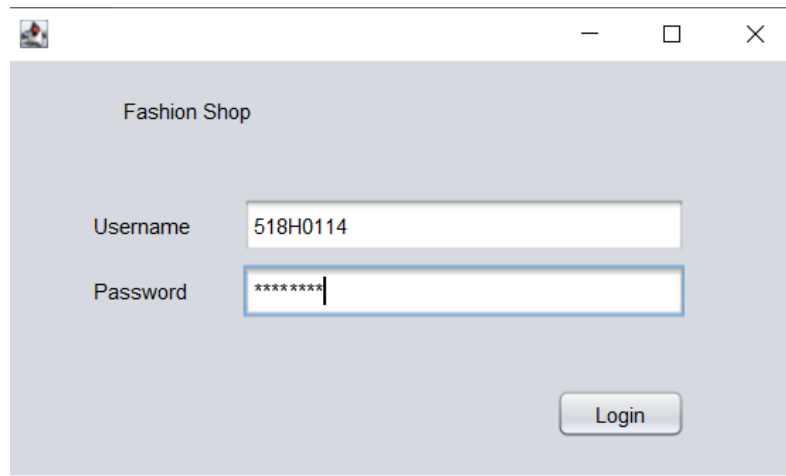
Link project được đăng lên github:

[GitHub - andyphan0508/518H0616_518H0114_DP](https://github.com/andyphan0508/518H0616_518H0114_DP)

PHẦN 4: DEMO

Đầu tiên là giao diện phần login. Có ba tài khoản để đăng nhập vào hệ thống:

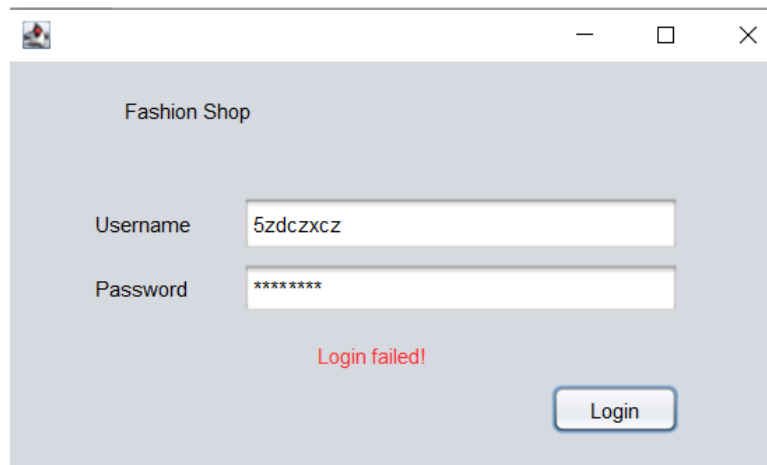
1. Username: 518H0114 – Password: 518H0114 => tên người dùng là Lê Tấn Tài
2. Username: 518H0616 – Password: 518H0616 => tên người dùng là Phan An Duy
3. Username: admin – Password: admin => tên người dùng là Admin



The screenshot shows a window titled "Fashion Shop" with a login form. The form has two input fields: "Username" and "Password". The "Username" field contains the text "518H0114". The "Password" field contains seven asterisks "*****". Below the fields is a "Login" button.

Hình 4.1 Giao diện phần login

Khi đăng nhập sai User name hoặc Password thì hệ thống sẽ báo lỗi



The screenshot shows the same "Fashion Shop" login window. The "Username" field now contains "5zdczxcz" and the "Password" field contains "*****". Below the fields, the text "Login failed!" is displayed in red. The "Login" button is still present.

Hình 4.2 Giao diện khi đăng nhập sai

Giao diện sau khi đăng nhập sẽ có các chức năng cơ bản như Tạo đơn hàng, xóa đơn hàng, tìm đơn hàng, thêm sản phẩm và sẽ có một bảng danh sách các đơn hàng được tạo thành công.

SHOP MANAGEMENT

User ID: 518H0114
Name: Le Tan Tai

Log out

Bills Management

Create Bill Add Item... Bill ID Find

No.	Bill ID	Price	Type	Creator
1	1	1000000	Direct	Le Tan Tai
2	2	2500000	Direct	Phan An Duy

Hình 4.3 Giao diện sau khi đăng nhập

Khi tạo đơn hàng nhân viên sẽ scan hoặc nhập ID của món hàng, nhập số lượng và chọn hình thức thanh toán (tiền mặt/thẻ/COD).

Creator: 518H0114 - Le Tan Tai Total: 750000 Save

Date: 2021-04-23

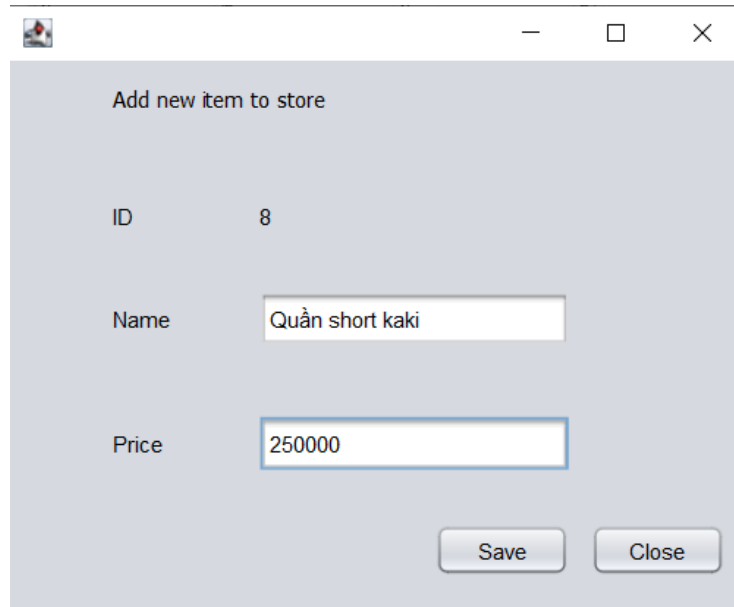
Payment: Cash

Item ID: 5 Quantity: 3 Add

No.	ID	Name	Price
1	5	Áo khoác jean	250000

Hình 4.4 Giao diện khi nhân viên tạo một đơn hàng mới

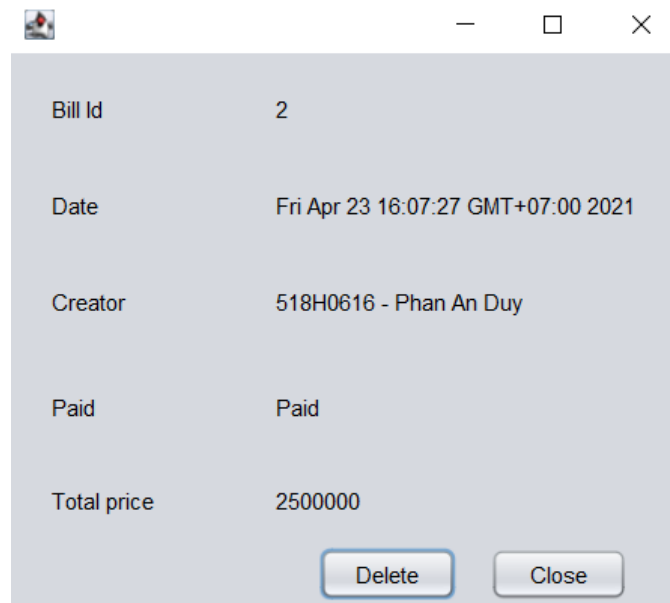
Chủ cửa hàng có thể thêm thông tin mặt hàng mới, ID sẽ được tạo tự động



A screenshot of a software window titled "Add new item to store". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is light gray and contains three labeled input fields: "ID" with the value "8", "Name" with the text "Quần short kaki", and "Price" with the value "250000". At the bottom right of the window, there are two buttons: "Save" and "Close".

Hình 4.5 Giao diện thêm một món hàng mới

Ngoài ra, người dùng có thể tìm kiếm thông tin đơn hàng và xóa đơn hàng bằng cách tìm kiếm qua ID của hóa đơn



A screenshot of a software window displaying bill details. The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is light gray and contains five labeled text fields: "Bill Id" with the value "2", "Date" with the value "Fri Apr 23 16:07:27 GMT+07:00 2021", "Creator" with the value "518H0616 - Phan An Duy", "Paid" with the value "Paid", and "Total price" with the value "2500000". At the bottom right of the window, there are two buttons: "Delete" and "Close".

Hình 4.6 Giao diện chi tiết đơn hàng

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Coder, G., 2021. Hướng dẫn Java Design Pattern - Singleton - GP Coder (Lập trình Java). [online] GP Coder. Available at: <<https://gpcoder.com/4190-huong-dan-java-design-pattern-singleton/>> [Accessed 14 April 2021].
2. Coder, G., 2021. Hướng dẫn Java Design Pattern - Factory Method - GP Coder (Lập trình Java). [online] GP Coder. Available at: <<https://gpcoder.com/4352-huong-dan-java-design-pattern-factory-method/>> [Accessed 14 April 2021].
3. Coder, G., 2021. Hướng dẫn Java Design Pattern – Observer - GP Coder (Lập trình Java). [online] GP Coder. Available at: <<https://gpcoder.com/4747-huong-dan-java-design-pattern-observer/>> [Accessed 14 April 2021].
4. Coder, G., 2021. Hướng dẫn Java Design Pattern – Template Method - GP Coder (Lập trình Java). [online] GP Coder. Available at: <<https://gpcoder.com/4810-huong-dan-java-design-pattern-template-method/>> [Accessed 14 April 2021].
5. Coder, G., 2021. Hướng dẫn Java Design Pattern – MVC - GP Coder (Lập trình Java). [online] GP Coder. Available at: <<https://gpcoder.com/5160-huong-dan-java-design-pattern-mvc/>> [Accessed 14 April 2021].
6. Sites.google.com. 2021. Kéo thả với NetBean Designer - OOP - JAVA. [online] Available at: <<https://sites.google.com/site/ptitooop/lab/lab4/netbean-designer>> [Accessed 17 April 2021].

Tiếng Anh

7. Design Patterns | Object Oriented Design. 2021. Design Patterns | Object Oriented Design. [online] Available at: <<https://www.oodesign.com/>> [Accessed 14 April 2021].
8. Tutorialspoint.com. 2021. Design Patterns in Java Tutorial - Tutorialspoint. [online] Available at: <https://www.tutorialspoint.com/design_pattern/index.htm> [Accessed 15 April 2021].

9. Các file bài giảng của thầy Nguyễn Thanh Phước