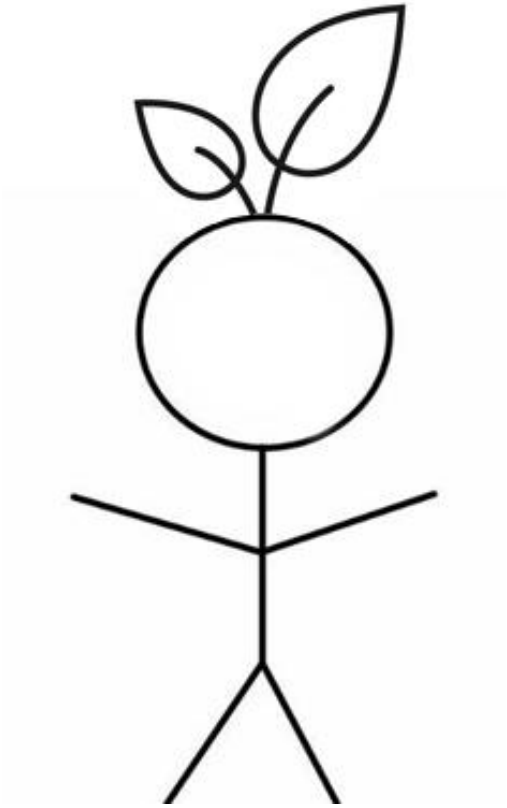
Growing Pains – An Online Plant Store System (OPSS)

Object Oriented Software Development - Project



By: Mark Lambert (C00192497)

Course: Software Development (CW_KCSOF_B)

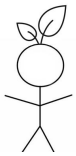
Submission Date: 11th April 2025

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Table of Contents

Page

Table of Figures



Summary

A rapidly growing houseplant store wants to expand its business to build an Online Plant Store System (OPSS) to manage its expanding business and improve customer engagement. The system should aim to meet the following requirements:–

a) Facilitate the buying of a diverse range of plants and plant accessories

b) A comprehensive marketplace experience, making the interface accessible to the user

c) A responsive, dynamic application that responds to user inputs and updates the backend database as the user interacts with the application

d) The store catalogue must have a filter feature to enhance user experience, allowing users to sort items and accessories by price, type etc

e) The user must be able to edit their account information as well as view and cancel any orders made

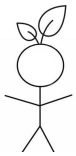
f) Users must also be able to set personal reminders, notifying them of when to water their plants

The system should also feature a user-interface which keeps the design aesthetics of the houseplant store in mind.

The system should allow users to browse the store catalogue, which will include filtering options such as plant species, type, price and accessory. Each plant listing will include brief descriptions, care instructions and pricing information, which can be visible when a customer selects a plant.

Once the user selects a plant, they can add it to their cart to proceed with the checkout process. Users may update their cart or remove items. When the user initiates the checkout process, they must enter in payment details before finally placing the order.

Another integrated feature should allow users to set a reminder by selecting a date. The reminder will take input on the plant type and species (e.g., succulent, tropical, houseplant) and notify the user of when to next water their plant.



Requirements

The following document outlines the requirements for the Online Plant Store System (OPSS). To ensure that all corners of the requirements finding process were covered, the **FURPS+** model to assess functional and on-functional requirements was considered.

Functional

Requirement ID :	**FR001:** Select Item
Definition:	The system shall display detailed information when a user selects an item from the catalogue.
Specification:	- On click of a product in the BrowsePanel the system should: - Form a SELECT query on the Product table - Display a new ProductPanel with the following using JLabel's: 200x200 image of the product, product name, price and description

Requirement ID :	**FR002:** Cart Management
Definition:	Registered users may add products to their cart, from which they may alter the product quantity via a JSpinner
Specification:	- On click of the "Add to Cart" button the system must: - Validate the user is first logged in - Update the users cart object to display the product quantities and total price

Requirement ID :	**FR003:** Checkout Process
Definition:	Users complete the Order by initiating a checkout process, validated by inputting payment details
Specification:	- On click of the "Checkout" button the system must: - Build a form to input: Card Number, Card Holder, Address, CVV and Expiration Date (via JComboBox's) - On submit, the system will generate an INSERT query into the Orders table

Requirement ID :	**FR004:** Browse Catalogue
Definition:	The system shall display a populated catalogue of items with a scrollable UI
Specification:	- A JPanel displaying a series of product item containers which hold information about each product in the Product table. - Products are retrieved via a SELECT query in the Products table

Requirement ID :	**FR005:** Order History
Definition:	Users must be able to view past orders with the aim of cancelling orders should they wish
Specification:	- A JTable displaying a history of all orders made by the logged in user. - The table is populated via a SELECT query on the Customer table which INNER JOINS with the Orders table - When an order is selected, the user may cancel the order by clicking the “Cancel Order” button. - Onclick, a DELETE query in the Orders table is generated

Non-Functional

Requirement ID :	**NFR001:** Usability
Definition:	The system must be both learnable and accessible for new users
Specification:	- Learnability: Users must be able to comfortably adapt to the systems GUI, enabling them to purchase products quickly - Accessible: The system must be designed bearing in mind users who may have vision impairments, such as font sizes, colours etc. - Logging errors to a standard text file is a must, ensuring users can learn in more detail what errors may have occurred

Requirement ID :	**NFR002:** Reliability
Definition:	The system must reliably deal with invalid data input from the user
Specification:	- Data input must be handled appropriately according to what may constitute as “bad data” or malicious data. - Preventative measures against SQL Injection by using prepared statements

Requirement ID :	**NFR003:** Performance
Definition:	The system must respond quickly and appropriately to user input
Specification:	- Interaction between the system and database must be seamless, ensuring the customer is met with a responsive application - Any image scaling must be handled appropriately and with care, to ensure the performance drawback is not apparent to the user

Requirement ID :	**NFR004:** Supportability
Definition:	The system must be maintainable for future iterations and expansion
Specification:	- Code must be well documented and conform to standard Object-Oriented principles - System architecture must be well organised and make use of a MVC structure - Extensive version history must be available on a version control platform

Requirement ID :	**NFR005:** Security
Definition:	The system must be secure for the user to use
Specification:	- Any sensitive or precious data shall be handled with care - Passwords must be securely stored in the database by first hashing the input. A standard SHA-256 algorithm may be deployed.

Database Tables

Customer Table

```
mysql> desc Customer;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| customerID | int         | NO   | PRI | NULL    | auto_increment |
| fName      | varchar(35) | YES  |     | NULL    |                |
| lName      | varchar(50) | YES  |     | NULL    |                |
| email      | varchar(50) | YES  |     | NULL    |                |
| address    | text        | YES  |     | NULL    |                |
| password   | varchar(64) | YES  |     | NULL    |                |
| phone      | varchar(15) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Figure 1: Structure of Customer Table

```
+------------+------------+---------+--------------------------+--------------------------------+-----------
--+
| customerID | fName      | lName   | email                    | address                        | password
  |
+------------+------------+---------+--------------------------+--------------------------------+-----------
--+
|          1 | Aoife      | Murphy  | aoife.murphy@gmail.com   | 12 Main St, Dublin, Ireland    | de9ad42a71
9 |
|          2 | Sean       | Murphy  | sean.oconnor@hotmail.ie  | 45 Elm Road, Cork, Ireland     | e337af8bad
  |
|          3 | Padraig    | Kelly   | padraig.kelly@example.com| 34 Pine Lane, Limerick, Ireland| 966e2eaa40
  |
|          4 | Mark       | Lambert | marklambert123@gmail.com | Ireland                        | 9390298f3f
  |
|          5 | Hannah     | Flint   | flinthannah@aol.com      | Goldthorpe, Yorkshire, England | 46b9e6bdf3
2 |
|         16 | Ad         | Min     | admin@growingpains.com   | Admins Basement                | ca978112ca
  |
|         17 | Password = | a       | a@a                      | Note the hash output for "a"   | ca978112ca
  |
|         18 | Boe        | Jloggs  | boejloggs@geocities.com  | 308 Negra Arroyo Lane          | 0206f2eade
  |
+------------+------------+---------+--------------------------+--------------------------------+-----------
```

Figure 2: Sample Data of Customer Table

[1]

[1] Note, password field has been cropped for readability reasons, as the hashed password stretches the screenshot aspect ratio out, making text too small

Product Table

mysql> describe Product;

Field	Type	Null	Key	Default	Extra
productID	int	NO	PRI	NULL	auto_increment
productName	varchar(40)	YES		NULL	
description	text	YES		NULL	
price	decimal(4,2)	YES		NULL	
qty	int	YES		NULL	
category	varchar(30)	YES		NULL	
image_path	varchar(255)	NO		NULL	

Figure 3: Structure of Product table

[2]

mysql> select * from Product;

productID	productName	description	price	qty	category	image_path
3	Pothos	Pothos 13cm pot, suitable for all owners	14.99	25	Plant	images/pothos.png
4	Golden Mister	Golden mister, ideal for orchids and high hummidty plants	8.99	12	Accessory	images/mister.png
5	Monstera	Monstera Adasonii w/ 15cm pot, suitable for all owners	12.99	41	Plant	images/monsterra.png
6	7cm Giraffe Pot	Gooft Giraffe pot to make your plants more fun	2.99	23	Accessory	images/griaffe_pot.png
7	Maidenhair Fern	Maidenhair Fern w/ 6cm pot, suitable for all owners	6.99	4	Plant	images/fern.png
8	Green Pot	Green Pot w/ Eye Design	8.99	3012	Accessory	images/pot_eyes.png
21	Golden Pothos	Golden Pothos w/ 8cm pot, suitable for all owners	5.99	26	Plant	images/golden_pothos.png
22	13cm Pot with Motif	Hand painted ceramic pot	10.99	10	Accessory	images/pot_egg.png
23	Pilea	Chinese Money plant (Pilea) in a kokedama. Kokedama is a ball of soil, covered with moss on which an ornamental plant grows. The idea has its origins in Japan, where it is a method of bonsai styling. Watering: Mist regularly, immerse in water when dry	5.99	27	Plant	images/pilea.png
24	Moisture Meter	Moisture Meter - Single probe, excellent for all experience levels for watering	4.99	38	Accessory	images/moisture_meter.png
25	Spider Plant	Spider Plant - Suitable for all experience levels, loves humidity	9.99	27	Plant	images/spider.png
26	8cm Pot with Wooden Stand	Duck egg blue pot with wooden stand	14.99	4	Accessory	images/pot_stand.png
27	String of Hearts	String of Hearts - Vining indoor plant	9.99	12	Plant	images/soh.png
28	String of Bananas	String of Bananas - Succulent vining plant	19.99	10	Plant	images/sob.png
29	Alocasia Poly	Alocasia - Elephant's Ear or Poly	15.99	14	Plant	images/alocasia.png

Figure 4: Sample Data for Product table

[2] Note, the description for productid 23 is intentionally long-winded to demonstrate usage of JTextArea elements.

Orders

```
mysql> describe Orders;
+-----------------+---------------+------+-----+---------+----------------+
| Field           | Type          | Null | Key | Default | Extra          |
+-----------------+---------------+------+-----+---------+----------------+
| orderID         | int           | NO   | PRI | NULL    | auto_increment |
| customerID      | int           | YES  |     | NULL    |                |
| date            | date          | YES  |     | NULL    |                |
| time            | time          | YES  |     | NULL    |                |
| shippingAddress | text          | YES  |     | NULL    |                |
| totalPrice      | decimal(10,2) | YES  |     | NULL    |                |
+-----------------+---------------+------+-----+---------+----------------+
```

Figure 5: Structure of Orders Table

```
mysql> select * from Orders;
+---------+------------+------------+----------+-----------------+------------+
| orderID | customerID | date       | time     | shippingAddress | totalPrice |
+---------+------------+------------+----------+-----------------+------------+
|       1 |          4 | 2025-03-29 | 11:58:21 | Ireland         |      95.90 |
|       2 |          4 | 2025-03-29 | 20:41:34 | Ireland         |       8.99 |
|       3 |          4 | 2025-03-29 | 20:43:54 | Ireland         |       6.99 |
|       4 |         11 | 2025-03-29 | 21:18:40 | a               |      19.98 |
|       5 |         11 | 2025-03-29 | 23:49:18 | a               |       5.99 |
|       6 |         11 | 2025-03-30 | 00:10:52 | a               |       2.99 |
|       7 |          4 | 2025-03-30 | 00:23:27 | Ireland         |       0.00 |
|       8 |          4 | 2025-03-30 | 00:24:47 | Ireland         |       9.99 |
|       9 |          0 | 2025-03-30 | 17:35:41 | a               |      64.94 |
|      10 |         10 | 2025-03-30 | 17:37:29 | a               |       6.99 |
|      16 |         12 | 2025-03-30 | 22:53:41 | a               |      10.99 |
|      17 |         10 | 2025-04-02 | 00:10:16 | a               |      12.99 |
+---------+------------+------------+----------+-----------------+------------+
```

Figure 6: Sample Data for Orders Table

ER Diagram

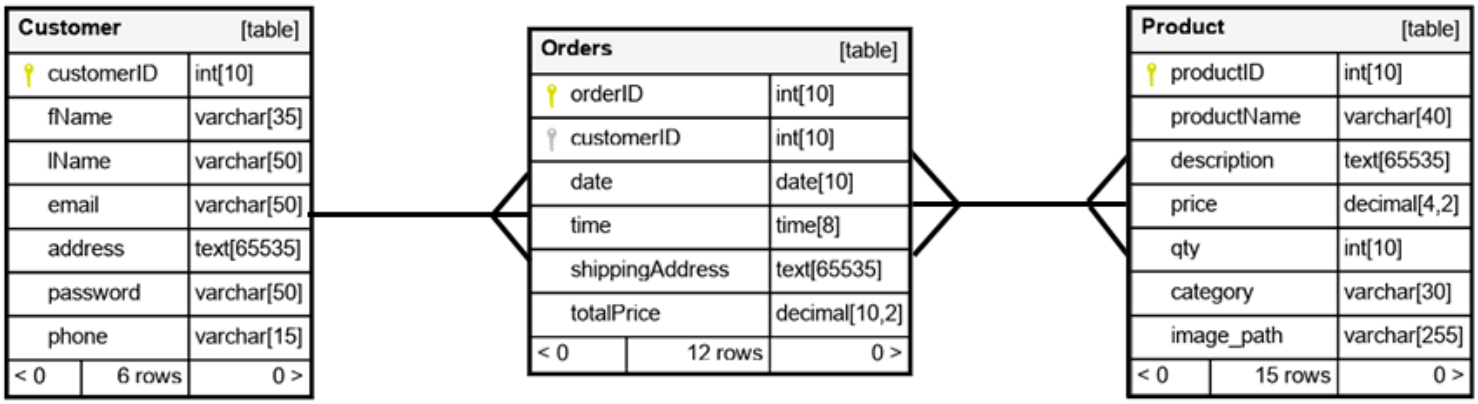


Figure 7: ER Diagram for GrowingPains

[3]

[3] The Many to Many relationship between the Orders and Product tables in the current iteration of the OPSS has not been simplified to include an Order/Product table.

Future expansion during the Summer of '25 will ensure this implementation is appropriately handled.

Interesting Source Code Snippets

Retrieval by List

```
/**
 * Retrieves all Products from the Product table
 * @return A List of DisplayItem object's
 * @throws SQLException Error should a product not be found in the table
 */
public List<DisplayItem> getAllProducts() throws SQLException {
    ResultSet resultSet = null;
    ArrayList of products to store all selected products
    List<DisplayItem> products = new ArrayList<>();
    try {
        PreparedStatement pstat = connection.prepareStatement("SELECT productID, productName, description, price, qty, category, image_path FROM Product");
        //Assign resultSet the value of the query
        resultSet = pstat.executeQuery();

        Check if resultSet has a value
        while(resultSet.next()) {
            int id = resultSet.getInt("productID");
            String name = resultSet.getString("productName");
            String desc = resultSet.getString("description");
            double price = resultSet.getDouble("price");
            int qty = resultSet.getInt("qty");
            String image_path = resultSet.getString("image_path");

            DisplayItem product = new DisplayItem(id, name, desc, price, qty, image_path);
            products.add(product);
            }
        }
        catch(SQLException sqlException) {
            System.err.println("Error retrieving all Products from table : " + sqlException.getMessage());
            sqlException.printStackTrace();
        }

    return products;
}
```

Figure 8: getAllProducts() - ProductCrud

Lists are commonly used throughout the system, particularly when performing retrieval operations on the Product table. As seen in Figure 8- it highlights this retrieval process, where:

- The result of the select query is encapsulated within a DisplayItem object.
- Each DisplayItem gets added to a List of DisplayItem's, where each entry in the database is a DisplayItem, and each field is encapsulated within said item.
- The method returns the full list of products.

Displaying a List of Objects

```
public void getProducts(List<DisplayItem> products, ProductPanel p) {

        For each Item in the List of Items
    for (Item product : products){
        String image_path = product.getImgPath();
        ImageIcon icon = new ImageIcon(getClass().getResource(image_path));
        Image img = icon.getImage().getScaledInstance(250,  250,  Image.SCALE_SMOOTH);
        JLabel imgLabel = new JLabel(new ImageIcon(img));

        JLabel nameLabel = new JLabel(product.getItemName(), SwingConstants.CENTER);
        JLabel priceLabel = new JLabel("€" + product.getPrice(), SwingConstants.CENTER);
        priceLabel.setFont(PRODUCTFONT);
        nameLabel.setFont(new Font("Arial", Font.BOLD, 24));
        //Place each product into its own container, as we want to display the image,
        //product name and price
        JPanel productPanel = new JPanel(new BorderLayout());
        JPanel infoPanel = new JPanel(new BorderLayout());

            Container just for the text info
        infoPanel.add(nameLabel, BorderLayout.NORTH);
        infoPanel.add(priceLabel, BorderLayout.CENTER);

            Container for the image AND the text
        productPanel.add(imgLabel, BorderLayout.NORTH);
        productPanel.add(infoPanel, BorderLayout.CENTER);

        gridPanel.add(productPanel);
    }
```

Figure 9: getProducts - BrowsePanel

Figure 9 demonstrates how to display the List of DisplayItem's retrieved from Figure 8

- Firstly, a **for each** loop is utilised. In this case, for each Item in the DisplayItem list (DisplayItem extends the abstract class Item), extract the information from the DisplayItem
- Create a new JPanel container for each DisplayItem which holds the extracted information.
- The container makes use of a **BorderLayout,** allowing the individual pieces of product information to be displayed nicely within the container.
- Each container is added to the main JPanel **gridPanel**, which is the container for the **BrowsePanel** content area.

```
    Add event for clicking on the image of a product, so user can visit a detailed screen of that product
imgLabel.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
            //Set the title of the product to that of the currently selected product by calling product.getItemName
        p.setTitle(product.getItemName());
        p.setImage(product.getImgPath());
        p.setDescription(product.getDescription());
        p.setPrice(product.getPrice());
        p.setItem(product);
        Once the title is set, switch the cardLayout to the "Procuct" card, with the new heading matching the produc
        CONTROL.handleSelectedProduct();
    }

});
```

Figure 10: getProducts - BrowsePanel

Figure 10 displays how an event is handled between the user's **mouse** and the **imgLabel** associated with a Product in the **BrowsePanel.**

The method mouseClicked gets triggered when the users clicks the imgLabel. On click the **ProductPanel** p – representing **the selected items detailed panel,** gets assigned all the GUI elements associated with the selected product.

The final piece of code displays the relationship between a view packaged class, which handles any GUI based logic, and a controller packaged class, which handless the more business end logic concerned with the GUI interface. In this instance **CONTROL** represents the final class level variable of type **BrowseControl**. BrowseControl is a custom controller class which contains the necessary functions to operate on the Browse section of the system.

The purpose of diversifying the two packages is to provide an easier platform for expansion in the future, as the abstraction makes it much easier to change information and keeps the system modular.

Filtering a Catalogue of Products

```
/**
 * Filters products in the ProductPanel by querying the database based on the value of the filterList selected item
 * @param p The current panel displaying the current state of catalogue
 */
public void handleFilter(ProductPanel p) {
filterBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            //CLear grid
            gridPanel.removeAll();
            //Get new products, passing the string of the selected item to the controller argument
            getProducts(CONTROL.filterCatalogue(filterList.getSelectedItem().toString()), p);
        } catch (SQLException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

        //Revalidate screen
        revalidate();
        repaint();
        }

    });
}
```

Figure 11: handleFilter() - BrowsePanel

Figure 11 details the process of filtering the catalogue of items:

- A **JComboBox** is used to store the filters, which in this case are “Plant” and “Accessory” which are represented as “category” in the Product table.
- When the user selects a filter from the list and clicks the JButton **filterBtn,** a ActionEvent is triggered, calling **removeAll()** method on the gridPanel, removing all components from the container.
- **getProducts()** is then called with the argument **CONTROL.filterCatalogue(….)**
- This returns a list of items retrieved by querying the Product table based on the “category=?” where ? = the selected item from the JComboBox filterList.
- Finally, to dynamically display this new catalogue to the user, the revalidate() and repaint() methods are called.

Hashing Passwords

```java
public class PasswordHasher {

    /**
     * Performs thee SHA-256 hashing algorithm on the users input password
     * @param password The password input by the user
     * @return The hexadecimal of the function, converted to a string
     * @throws Exception thrown if an error occurs when attempting to perform the SHA-256 algorithm
     */
    public static String hashPassword(String password) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(password.getBytes(StandardCharsets.UTF_8));

        StringBuilder hexString = new StringBuilder();
        for (byte b : hash) {
            hexString.append(String.format("%02x", b));
        }

        return hexString.toString();
    }
```

Figure 12: Hashing

MessageDigest allows us to make use of the functionality of the SHA-256 algorithm. **Hashing** then follows the following steps:

- Convert the password string into an array of bytes
- The output of that array of bytes gets supplied to the argument **digest,** which finalises the hashing process by performing operations such as padding.
- The result of that function, is then ran through a for each loop, for each byte in the result, convert it back to a string, formatting each byte as a hexadecimal number

Writing to an Error File

```
            catch (ValidationException e4)
            {
                handleError(e4, "Validation Error");
//              e4.printStackTrace();
            }

        }
    });
}

/**
 * Helper variable used to display error messages to the user via a JOptionPane and write the error to an error file
 * @param errorType The type of error that occured
 * @param e The Exception occured
 */
private void handleError(Exception e, String errorType) {
    JOptionPane.showMessageDialog(CreateAccountPanel.this, e.getMessage(), errorType, JOptionPane.ERROR_MESSAGE);
    GrowingPains.errorWriter.logError(errorType, e.getMessage());
}
```

Figure 13: Error Writing

Figure 13shows a sample throw of a custom Exception **ValidationException**, which is thrown when a field is left blank in the account creation process. When the exception gets caught:

- The handleError method is invoked, with the Exception as an argument and a string describing the type of error.
- **handleError** creates a popup to alert the user of the error
- A static instance of an **ErrorWriter** object located in the GrowingPains class is then called with the logError() method to write to a file which is opened on system load.

Method to Store Images

	price	qty	category	image_path
	14.99	27	Plant	images/pothos.png
	8.99	13	Accessory	images/mister.png
	12.99	41	Plant	images/monsterra.png
	2.99	23	Accessory	images/griaffe_pot.png
	6.99	4	Plant	images/fern.png
	8.99	3012	Accessory	images/pot_eyes.png
	5.99	26	Plant	images/golden_pothos.png
	10.99	10	Accessory	images/pot_egg.png
	5.99	27	Plant	images/pilea.png
or watering	4.99	38	Accessory	images/moisture_meter.png
	9.99	27	Plant	images/spider.png
	14.99	4	Accessory	images/pot_stand.png
	9.99	15	Plant	images/soh.png
	19.99	10	Plant	images/sob.png
	15.99	14	Plant	images/alocasia.png

mark@mark-MacBookPro: ~

Figure 14: image_path in Product table

Observing Figure 14, noting the image_path, each product contains the relative path to a .png image, which relates to the plant. The images are located within a folder named **images** within the **view** folder.

CardLayout as a Central Navigation Tool

```
EDIT ACCOUNT Button
editAccountBtn = createButton("Account","images/account.png");
sideBar.add(editAccountBtn);
editAccountBtn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            checkLoggedIn("Edit Account");
            //customer = login.getLoggedInCustomer(); // Get current customer

            // Create edit panel with current customer
            edit = new EditAccountPanel(customer);
            mainContent.add(edit, "Edit Account");
            cardLayout.show(mainContent, "Edit Account");

            // No need to get updated customer here - it will be updated in the panel
        } catch (UserNotLoggedInException e1) {
            handleError(e1);
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
});
```

Figure 15: CardLayout

Figure 15 outlines the relationship of a CardLayout manager being used to switch to a new JPanel.

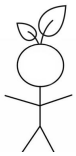
- The CardLayout works exactly like a deck of cards, where you can call the add() method to add a card to the deck, in this case the card being a JPanel container.
- A second argument is supplied to add(), a String indexing the added JPanel enabling you to easily reference the “card” when calling the show() method.
- The show() method brings the JPanel with the matching index to the foreground.

Driver and main() method

```
package controller;
//GROWING PAINS - A Plant Shop system
//Mark Lambert - C00192497 - Object Oriented Software Development 2 - Year 2 Semester 2

import java.sql.SQLException;
/**
 * Driver for GrowingPains application
 */
public class GrowingMains {

    /**
     * Main
     */
        public static void main(String[] args) {
            try {
                //New instance of GrowingPains application
                GrowingPains g = new GrowingPains();
                //Set GrowingPains to visible
                g.run();
            } catch (SQLException e) {
                e.printStackTrace();
            }

        }
}
```

Figure 16: Driver class

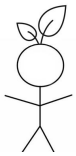
Figure 16 shows the benefits of the Object-Oriented approach to Software Development. The driver class for OPSS contains roughly 10 lines of code excluding comments, allowing for a much more manageable and extensible codebase for future iterations.



Test Cases

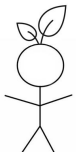
Name	**TC-001:** Select Item
Requirement	Verify that the system successfully updates to display details of a selected item from a database of items.
Preconditions	The system is displaying the full catalogue
Steps	1. Click on the **image** of the third item. 2. Return to previous page 3. Click on the **name** of the first item. 4. Return to previous page
Expected Results	1. The item details window appears, with a larger image and more detailed information 2. Verify that items can be selected by clicking icon **or** thumbnail 3. Return to browsing catalogue

Name	**TC-002:** Add to Cart
Requirement	Verify that the system successfully allows a user to enter item(s) to cart.
Preconditions	The user is viewing the catalogue
Steps	1. Click on the **second** product 2. Click the “Add to Cart” button 3. Return to catalogue 4. Click on the **fourth** product 5. Change **quantity** to 2, add to cart
Expected Results	1. The item details window appears, with a larger image and more detailed information 2. The system alerts the user to the fact that the item has been added successfully 3. Return to browsing catalogue



Name	**TC-003:** Checkout
Requirement	Verify that the store system successfully allows the user to checkout their items
Preconditions	The customer has items in their cart
Steps	1. Click on the **View Cart** button 2. Click on the **Proceed to Checkout** button 3. Enter your **login details** 4. Input **personal information** 5. Click on the **Confirm Order** button
Expected Results	1. The system displays the users Cart 2. The system begins the Checkout process 3. System prompts user for login details 4. System prompts user for shipping & billing information 5. System updates to confirm to the user that their order has been successfully placed

Name	**TC-004:** Filter Catalogue
Requirement	Verify that the system allows the user to apply filter(s) to the Catalogue of Items
Preconditions	The system is displaying the full catalogue
Steps	1. Click on the **Filter** button 2. Select **one** filter 3. Click the **Apply Filter** button 4. Click on the **Filter** button 5. Select **another** filter 6. Click on the **Apply Filter** button
Expected Results	1. The system updates to show a list of filters to choose from 2. When applied, the system displays just items matching the filter tag 3. Applying another filter will result in a more specific list of items



Name	**TC-005:** Schedule Reminder
Requirement	Verify that the system successfully sets and alerts the user when a Reminder is Scheduled
Preconditions	The user is logged in
Steps	1. Click on the **Schedule Reminder** button 2. Input **today's date** 3. Click on the **Set Reminder** button to confirm
Expected Results	1. The system will display the Schedule Reminder page 2. The system will update to display the reminder the user has just input

Name	**TC-006:** Browse Catalogue
Requirement	Verify that the system allows the user to browse the catalogue of items
Preconditions	The user is on the page displaying the catalogue
Steps	1. Click on the **Home** page 2. Scroll to browse the catalogue
Expected Results	1. The system will update in real time to display the catalogue – containing items - for the user