# **Interesting Source Code Snippets**

## A screenshot of a computer code AI-generated content may be incorrect.**Retrieval by List**

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.

## 

## A screenshot of a computer code AI-generated content may be incorrect.**Displaying a List of Objects**

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.

## A screen shot of a computer code AI-generated content may be incorrect. **Event Handling and View/Control Interaction**

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.

## A screen shot of a computer code AI-generated content may be incorrect.**Filtering a Catalogue of Products**

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.

## A computer screen shot of a computer code AI-generated content may be incorrect.**Hashing Passwords**

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

## A computer screen shot of a computer error AI-generated content may be incorrect.**Writing to an Error File**

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**

**A screen shot of a computer

AI-generated content may be incorrect.**

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.

## A screenshot of a computer code AI-generated content may be incorrect.**CardLayout as a Central Navigation Tool**

Figure 15: CardLayout

Figure 15outlines 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.

## A screenshot of a computer program AI-generated content may be incorrect.**Driver and main() method**

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.