Contribution to Developer Guide

Click here for the full document.

I covered the UI component for architecture section.

UI component

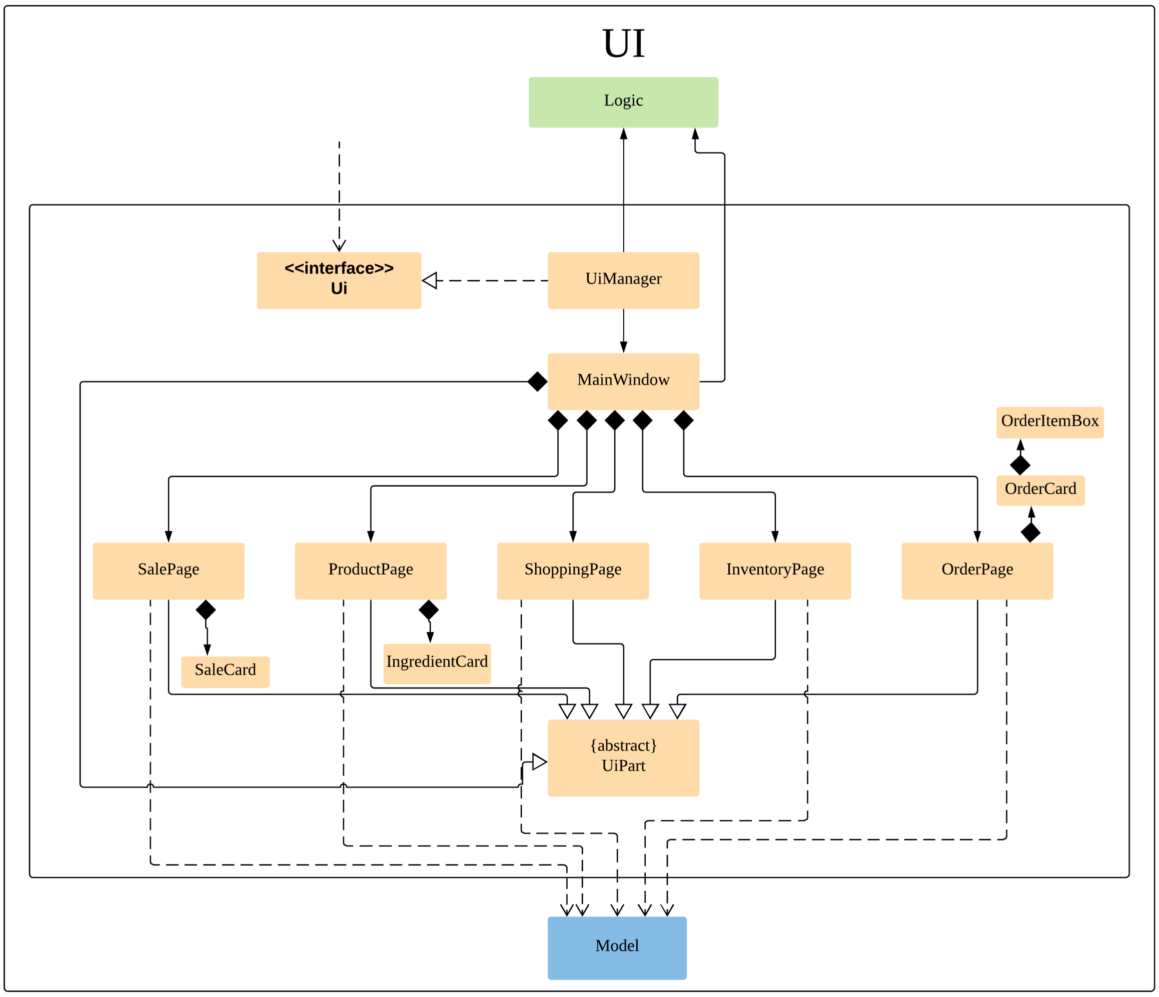

Figure 1. Structure of the UI Component

API : `Ui .java`

The UI consists of a `MainWindow` that contains 5 separate pages, namely the `SalePage`, `ProductPage`, `ShoppingPage`, `InventoryPage` and `OrderPage`. All these, including the `MainWindow`, inherit from the abstract `UiPart` class.

The `UI` component,

- Executes user commands using the `Logic` component.
- Listens for changes to `Model` data so that the UI can be updated with the modified data.

I wrote a use case to show the interaction between the user and the system.

Use case 6: Viewing Product’s ingredients

- **Precondition:** User has at least 1 product in the product list.
- **MSS**
  1. User can be viewing any page.
  2. User enters a ShowProductCommand indicating the index of the product to be shown, e.g. `product show 1`.
  3. BakingHome parses the command.
  4. BakingHome executes the command.
  5. BakingHome displays the details of the product.
- **Extensions**

  3a. BakingHome detects a invalid command.

  3a1. BakingHome shows an error message

  3a2. Use case ends.

Filter Product Feature

Implementation

BakingHome’s products can have two status: `ACTIVE` or `ARCHIVE`. This feature allows user to view products with a given certain status, i.e. shows only products with an ARCHIVE status.

The filter mechanism in product is facilitated by FilteredList which wraps a ObservableList and filters using the provided Predicate. A `FilteredList<Product> filteredProducts` is stored in the `ModelManager`. In `BakingHome`, there is an `ObservableList<Product> products` which contains all products, regardless of its status. `filteredProducts` in the ModelManager is initialized with this ObservableList.

Since a FilteredList needs a Predicate, which matches the elements in the source list that should be visible, the filter mechanism implements the following operation to support filtering:

- `Model#updateFilteredProductList(Predicate<Product> predicate)` — Sets the value of the property Predicate in the `filteredProducts`.
  - Predicates are declared statically in the `Model` interface, namely `PREDICATE_SHOW_ACTIVE_PRODUCTS`, `PREDICATE_SHOW_ARCHIVE_PRODUCTS`, and `PREDICATE_SHOW_ALL_PRODUCT`. In particular `PREDICATE_SHOW_ARCHIVE_PRODUCTS` is as follows

```
Predicate<Product> PREDICATE_SHOW_ARCHIVE_PRODUCTS = product -> {
    return product.getStatus() == Product.Status.ARCHIVE;
};
```

- The `FilterProductCommand` will call this method to change the visibility of products with different status by passing in the corresponding predicate.

An example usage scenario and how the filter mechanism behaves at each step is shown below.

Step 1. The user launches the application for the first time. `UniqueProductList` will be initialized with a list of default products in BakingHome. This list contains a few active products and a few archived products.

Step 2. The user inputs `product filter -scope archive` to list all archived products. `UI` passes the input to `Logic`. Logic then uses a few `Parser` classes to extract layers of information out.

...

Step 6. `Logic` gets the `FilterProductCommand` and execute it. The execution firstly calls `Model#updateFilteredProductList(Predicate<Product> predicate)` to update the Predicate in `filteredProducts` in `Model`. This execution then returns a `CommandResult` to `UI`, containing the response to the user.

Step 7. `UI` displays the response in the `CommandResult`. In addition, UI will change to display archived products after model updates `filteredProducts`, since `UI` is constantly listening for the change in `Model`.

The Sequence Diagram below shows how the components interact with each other for the above mentioned scenario.

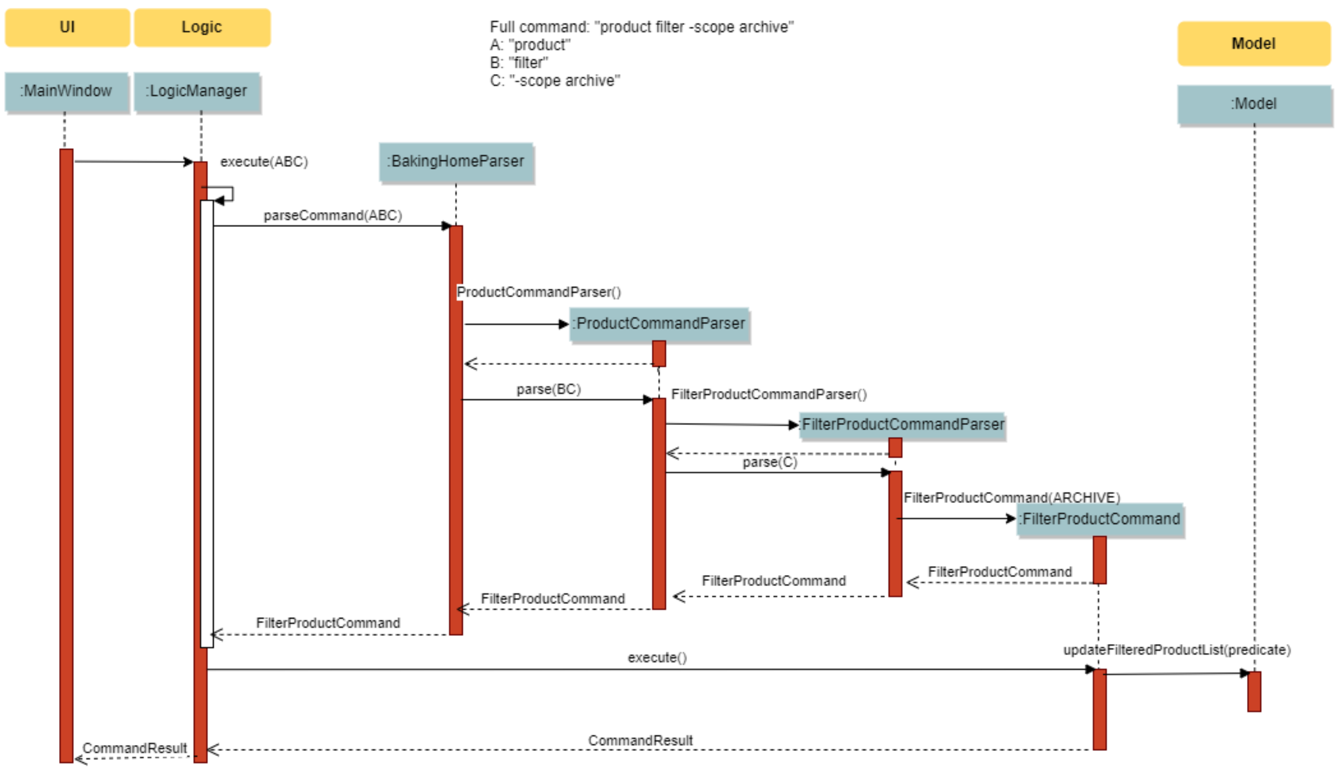

Figure 2. Sequence Diagram for Filter Product Mechanism

Note that almost all other commands follow the same sequence, with different `Command` and `Parser` classes.

Design considerations (excerpt)

- Alternative 2: Keep two separate product lists, one for archived products and one for active products.

- Advantages: Fast access to products of both status.
- Disadvantages: Implementation will become complicated. It also makes it very expensive when adding features like sorting all products according to name, price or cost.