Homework 12: Complex GUIs

Release date: 4/08
Due date: 4/15 by 11:59 PM

Goals

- Learn how to apply the MVC paradigm
- Learn how to handle events
- Learn how to add functionality to Swing components

Introduction

While working in the terminal for an eternity may be appealing to some, most users would like to interact with a graphical user interface, or GUI. GUIs, nowadays, can be found anywhere and everywhere — on our laptops, on our phones, at fast food locations, etcetera. They have made, and continue to make, computers easier to use for everyone.

The Java Programming Language offers built-in libraries to create graphical user interfaces, in the form of AWT and Swing, with Swing being the newer of the two. One of the nice things about these libraries is that they are cross-platform. If you create an application on your Mac, the same application can run seamlessly on your friend's PC. Below, you will use Swing to implement a small inventory tracking application.

Description

An IntelliJ project has been provided for you as skeleton code, and is accessible from the course webpage. This project contains classes and methods that have already been declared for you. All you need to do is add implementations where you find TODO comments in **`AddController`**, **`UpdateController`**, **`RemoveController`**, and **`SearchController`**. When you would like to run the application, you may do so by clicking the green arrow in the **`InventoryRunner`** class — this is where **`main()`** is declared. If you have any issues with setting this up in IntelliJ, please post on Piazza.

This application will act as an inventory tracker, and will allow the user to add, update, remove, and search for products. The **`Product`** and **`InventoryModel`** classes have been provided for you. **`Product`** is just a data holder that is used to store a product's SKU, name, wholesale price, retail price, and quantity. **`InventoryModel`** contains methods that interact with the products. You will be using these in your controller implementations. Below you will find specifications for what each controller should do.

AddController

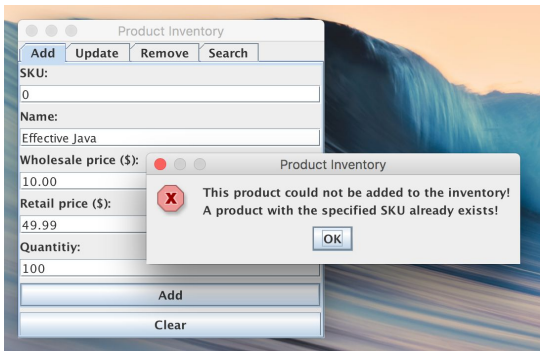
The **`AddController`** class defines the semantics of adding a new product to the inventory. Your implementations will go in the **`getAddButtonSemantics()`** and **`getClearButtonSemantics()`** methods. These methods are used in the constructor to add action listeners to the buttons.

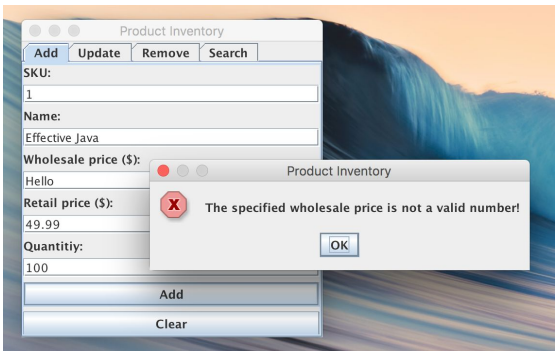
The user is presented with four **`JTextField`**s and two **`JButton`**s. The clear **`JButton`** should clear each **`JTextField`**, and request the focus of the SKU **`JTextField`**. You must perform input validation of the SKU, wholesale price, retail price, and quantity when the user clicks on the add **`JButton`**.

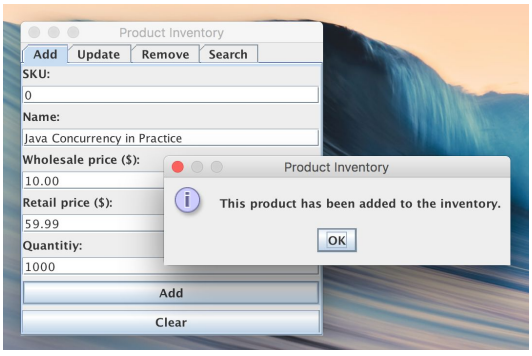
In order for an SKU to be valid, it must be unique. This means you must ensure that no existing product already has the specified SKU. If it is the case that the specified SKU is not unique, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the SKU **`JTextField`**.

In order for the wholesale price, retail price, and quantity to be valid, they must be numbers. The specified wholesale price and retail price can be a real number, but the quantity must be an integer. If any of the inputs are invalid, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of its **`JTextField`**.

If all inputs are valid, add the product to the **`InventoryModel`**, display a suitable success message using **`JOptionPane`** with a message type of **`INFORMATION_MESSAGE`**, and request the focus of the SKU **`JTextField`**.







UpdateController

The **`UpdateController`** class defines the semantics of updating the field value of an existing product in the inventory. Your implementations will go in the **`getUpdateButtonSemantics()`** and **`getClearButtonSemantics()`** methods. These methods are used in the constructor to add action listeners to the buttons.

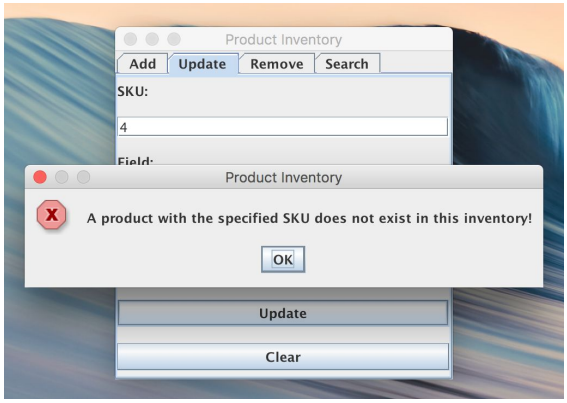
The user is presented with two **`JTextField`**s, one **`JComboBox`**, and two **`JButton`**s. The clear **`JButton`** should clear each **`JTextField`** and the **`JComboBox`**, then request the focus of the SKU **`JTextField`**. You must perform input validation of the SKU, field, and new field value when the user clicks on the update **`JButton`**.

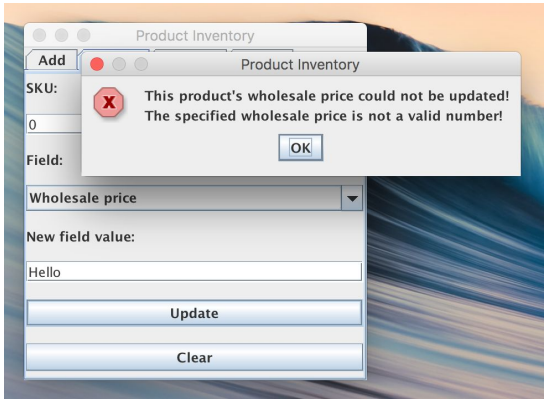
In order for the SKU to be valid, it must be associated with an existing product. If it is not, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the SKU **`JTextField`**.

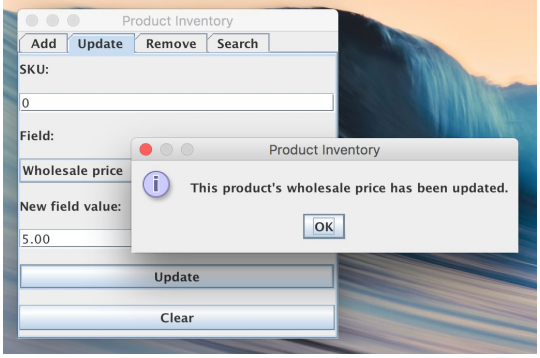
In order for the field to be valid, one must be selected from the **`JComboBox`**. If one is not, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the field **`JComboBox`**.

The validity of the new field value depends on the selected field. If the field is SKU, the new SKU must be unique. If the field is wholesale price, retail price, or quantity, the new value must be a valid number. If the new field value is not valid, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the new field **`JTextField`**.

If all inputs are valid, update the product in the **`InventoryModel`**, display a suitable success message using **`JOptionPane`** with a message type of **`INFORMATION_MESSAGE`**, and request the focus of the SKU **`JTextField`**.







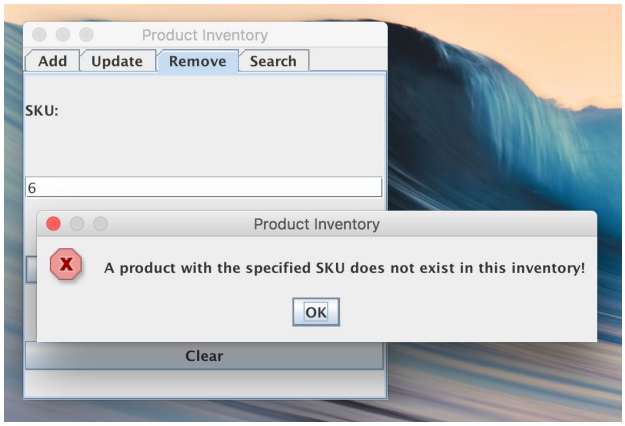
RemoveController

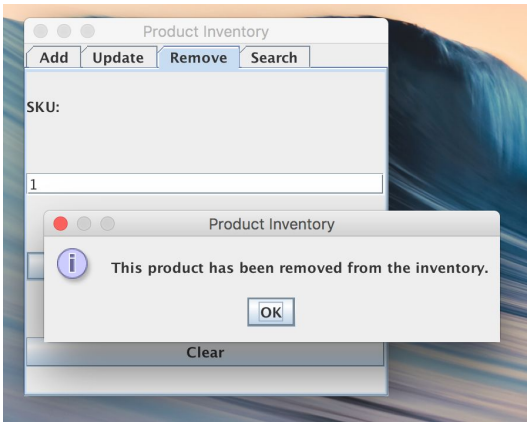
The **`RemoveController`** class defines the semantics of removing an existing product from the inventory. Your implementations will go in the **`getRemoveButtonSemantics()`** and **`getClearButtonSemantics()`** methods. These methods are used in the constructor to add action listeners to the buttons.

The user is presented with one **`JTextField`** and two **`JButtons`**. The clear **`JButton`** should clear the **`JTextField`**, then request the focus of it. You must perform input validation of the SKU.

In order for the SKU to be valid, it must be associated with an existing product. If it is not, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the SKU **`JTextField`**.

If the SKU is valid, remove the product from the **`InventoryModel`**, display a suitable success message using **`JOptionPane`** with a message type of **`INFORMATION_MESSAGE`**, and request the focus of the SKU **`JTextField`**.





SearchController

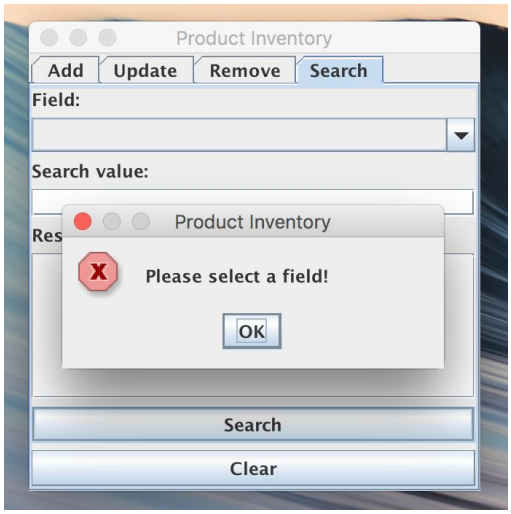
The **`SearchController`** class defines the semantics of searching for products in the inventory. Your implementations will go in the **`getSearchButtonSemantics()`** and **`getClearButtonSemantics()`** methods. These methods are used in the constructor to add action listeners to the buttons.

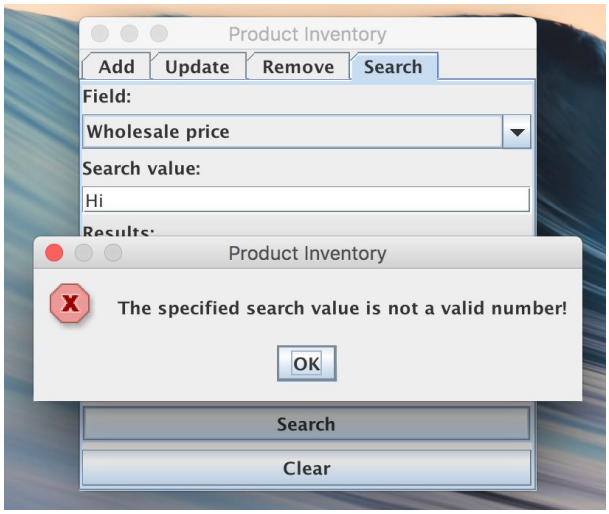
The user is presented with one **`JTextField`**, one **`JComboBox`**, one **`JTextArea`**, and two **`JButtons`**. The clear **`JButton`** should clear the **`JTextField`**, **`JComboBox`**, and **`JTextArea`**, then request the focus of it. You must perform input validation of the field and the search value.

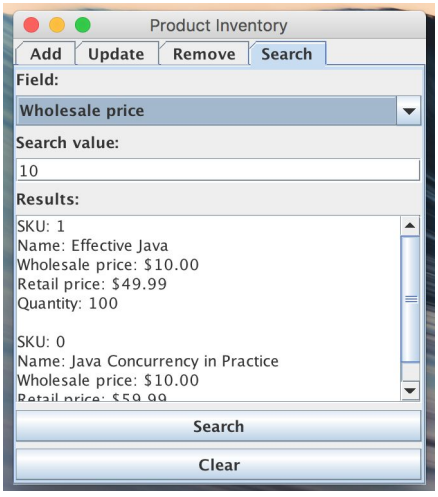
In order for the field to be valid, one must be selected from the **`JComboBox`**. If one is not, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the field **`JComboBox`**.

The validity of the search value depends on the selected field. If the field is wholesale price, retail price, or quantity, the search value must be a valid number. If the search value is not valid, provide a suitable error message using **`JOptionPane`** with a message type of **`ERROR_MESSAGE`**, and request the focus of the new field **`JTextField`**.

If all inputs are valid, search for the specified search value in the **`InventoryModel`**. The results should be displayed in the results **`JTextArea`**. You may use **`Product`**'s **`toString()`** method to easily get a **`String`** representation of each product. Then request the focus of the field **`JComboBox`**.







Submission

Submit **`AddController.java`**, **`UpdateController.java`**, **`RemoveController.java`**, and **`SearchController.java`** to Vocareum ***through Blackboard***.

We also ask that you submit a JAR file in order to made grading easier. One can be created with IntelliJ. In the menu, select **`Build`** > **`Build Artifacts`**… > **`homework-twelve:jar`** > **`Build`**. The newly created JAR will then be in the out/artifacts folder of the **`homework-twelve`** IntelliJ project. If you need any assistance with this, please post on Piazza.

Grading Rubric

- 25 points each (100 total)
  - **`AddController`**
  - **`UpdateController`**
  - **`RemoveController`**
  - **`SearchController`**