FACULTY OF ENGINEERING AND APPLIED SCIENCE

Software Design and Architecture SOFE 3650U

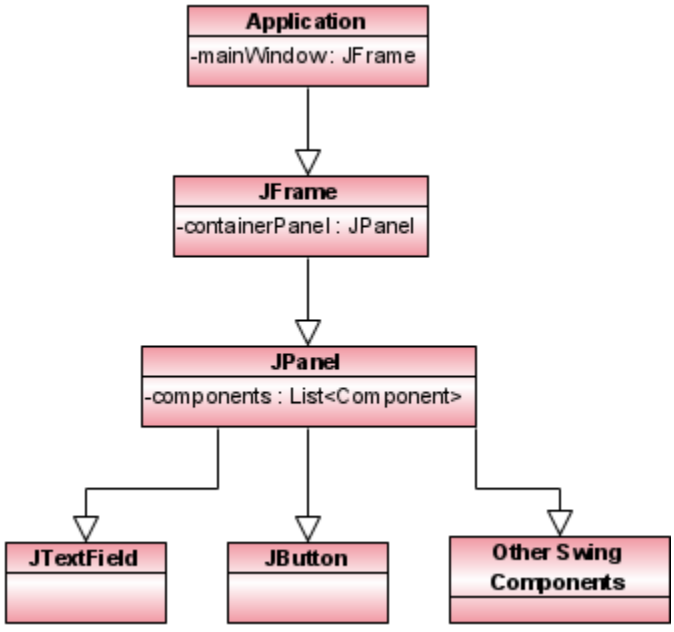
Assignment 2

Course Instructor	Harshvardhan Singh
Due Date	October 13, 2024
Group number	Group 8
Group Members	Mohammed Nasser - 100852276 Mahnoor Jamal - 100822030 Yordanos Keflinkiel - 100867864

Purpose of the Swing framework

The Swing framework is a crucial part of Java's Standard Library, specifically designed for creating graphical user interfaces (GUIs) that are rich in functionality and platform-independent. As a lightweight toolkit, Swing offers a wide array of pre-built components, including buttons (`JButton`), text fields (`JTextField`), labels, tables, and more, all of which can be customized and combined to develop responsive and interactive desktop applications. Unlike older GUI frameworks such as AWT (Abstract Window Toolkit), Swing is entirely written in Java, ensuring cross-platform compatibility without reliance on native operating system elements. In the diagram provided, the application's main window is represented by a `JFrame`, which acts as the top-level container for various components, while a `JPanel` inside the `JFrame` serves as a container to organize and manage multiple components like buttons, text fields, and others. These components are stored in a list and arranged according to the layout manager selected by the developer, allowing for a structured design. Overall, Swing's architecture promotes flexibility and reusability, empowering developers to create sophisticated, visually appealing user interfaces with minimal effort.

Class diagram of the Components of Swing



Implementation of the MVC Pattern

Differences from Conventional MVC Pattern

The implementation in the Swing example might differ from the conventional MVC pattern described in lectures in several ways:

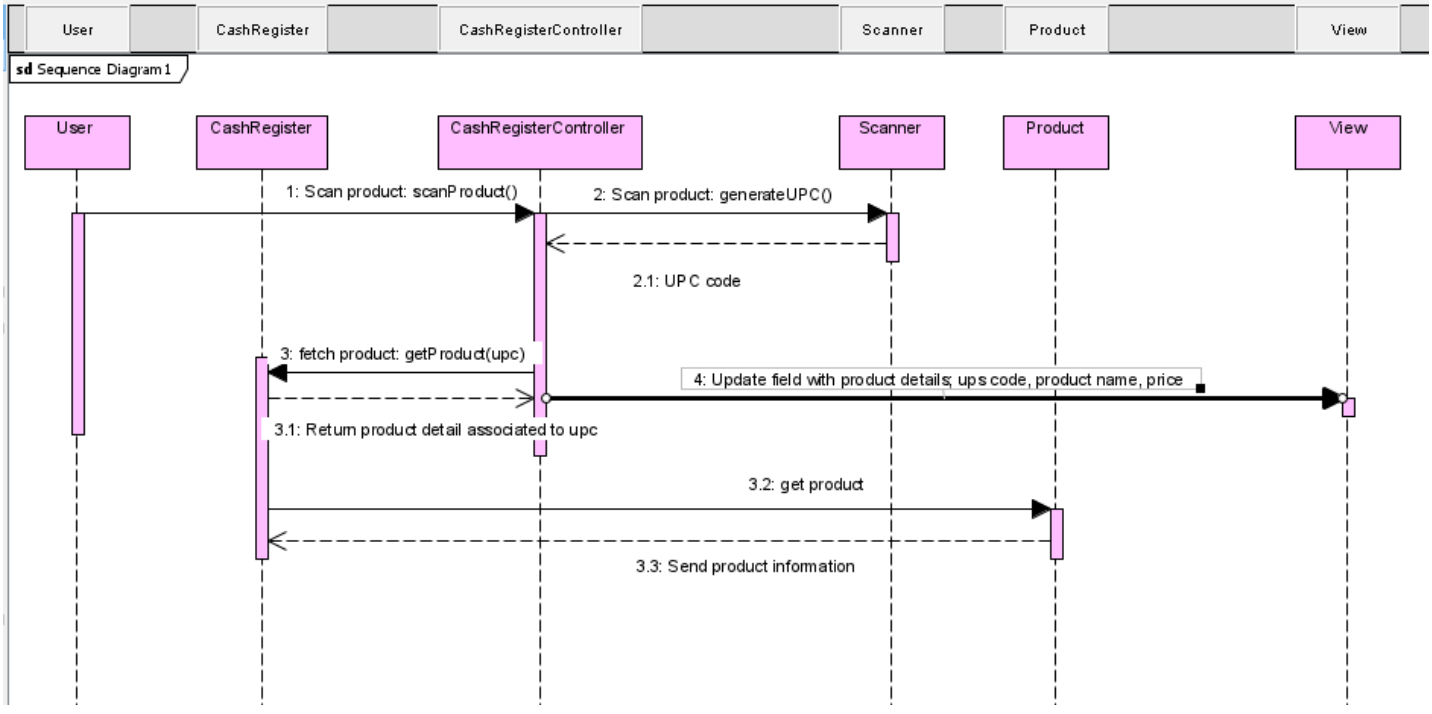
1. **Integration of View and Controller:** In many Swing applications, the View and Controller are intimately linked. It is common for the View classes to declare and use event listeners directly, as they are essentially a component of the Controller. The Controller would be a different class that handles all user inputs and logic in contrast to a more stringent separation.
2. **Flexible Role of Components:** Swing gives the MVC components adaptable roles. A single Swing class, for example, may serve as both a Model and a View, or a View and a Controller, particularly in smaller applications. Variations from the rigorous theoretical MVC architecture, in which each component is precisely and strictly defined, may result from this flexibility.
3. **Dynamic Updates:** Swing's architecture enables dynamic updates of View components straight from the Model or Controller, eliminating the need for a full view refresh. While this can provide more fluid interactions, it also departs from traditional MVC, in which the Controller would rigorously control all updates to the View through specified interfaces.

In summary, while the example follows the general principles of MVC, it adapts the pattern to fit the needs and capabilities of the Swing framework, which encourages a more integrated approach to handling user interface logic and data management.

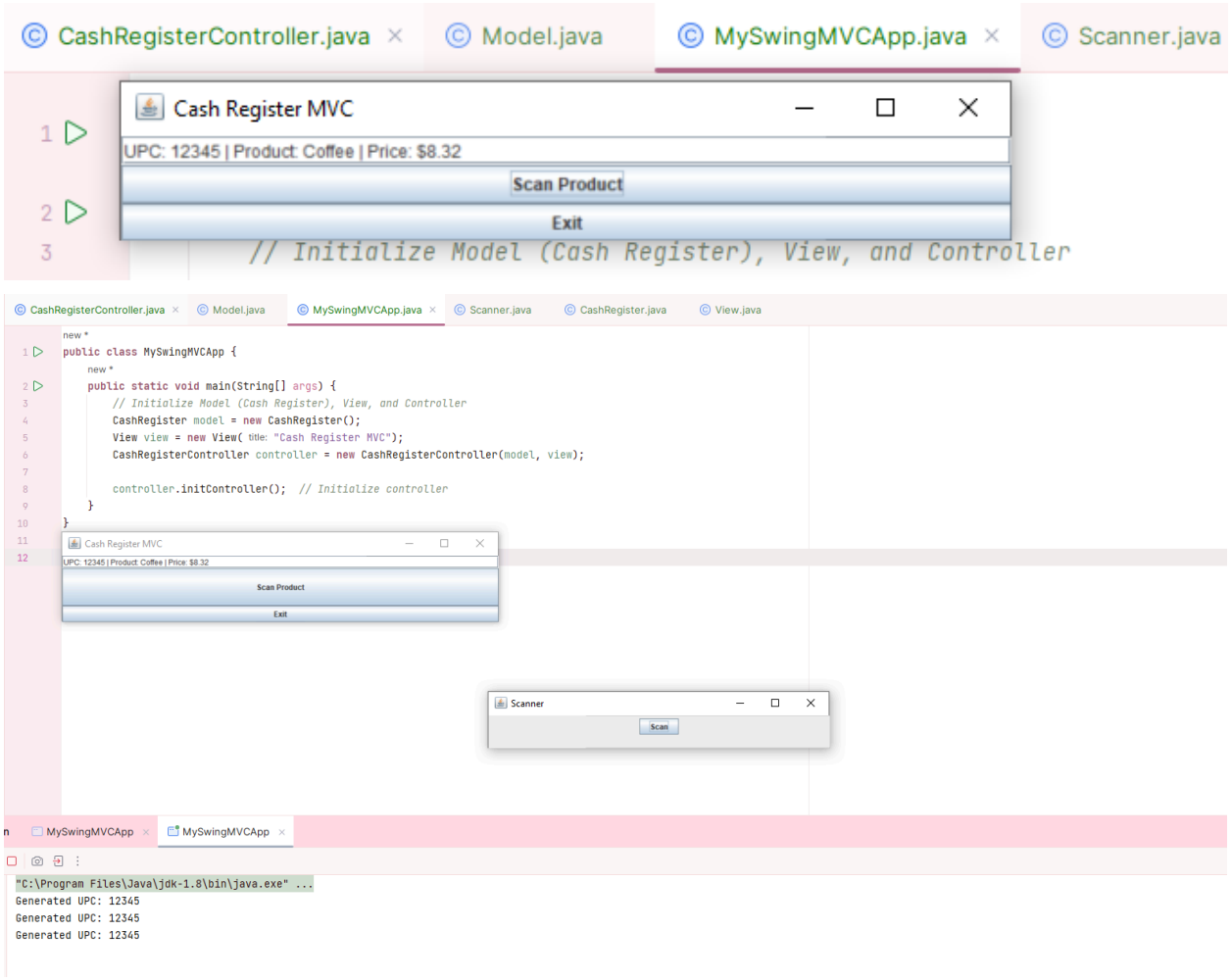
Cash Register Implementation Using Swing MVC Model

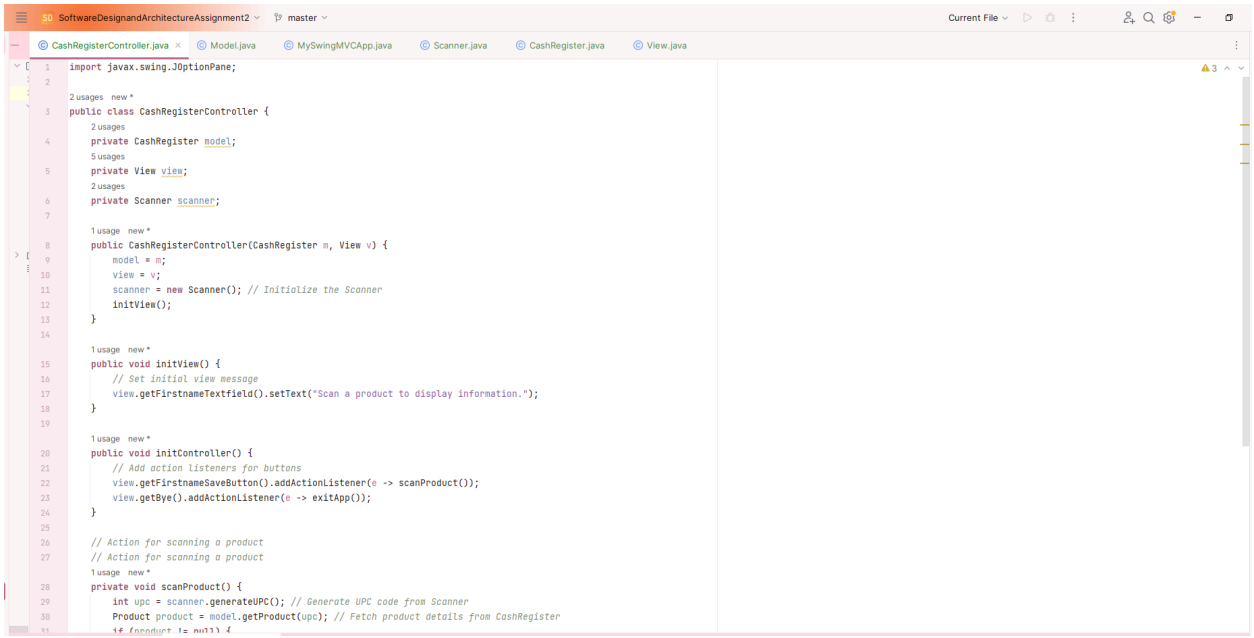
Code screenshots were uploaded to GitHub, Canvas, and screenshots can be found at the end of this document.

Cash Register Sequence Diagram



Output and Code:



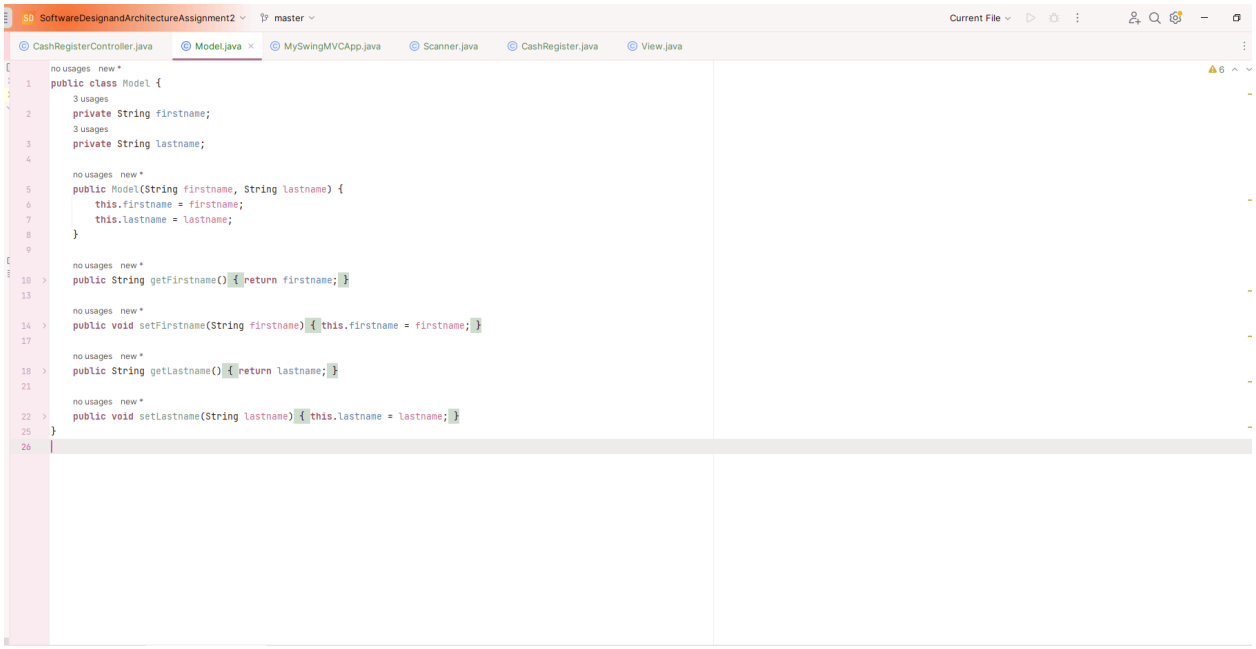

SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
import javax.swing.JOptionPane;
2 usages new *
public class CashRegisterController {
2 usages
private CashRegister model;
5 usages
private View view;
2 usages
private Scanner scanner;
1 usage new *
public CashRegisterController(CashRegister m, View v) {
model = m;
view = v;
scanner = new Scanner(); // Initialize the Scanner
initView();
}
1 usage new *
public void initView() {
// Set initial view message
view.getFirstnameTextfield().setText("Scan a product to display information.");
}
1 usage new *
public void initController() {
// Add action listeners for buttons
view.getFirstnameSaveButton().addActionListener(e -> scanProduct());
view.getBye().addActionListener(e -> exitApp());
}
// Action for scanning a product
// Action for scanning a product
1 usage new *
private void scanProduct() {
int upc = scanner.generateUPC(); // Generate UPC code from Scanner
Product product = model.getProduct(upc); // Fetch product details from CashRegister

```
    1 usage  new *
    public void initController() {
        // Add action listeners for buttons
        view.getFirstnameSaveButton().addActionListener(e -> scanProduct());
        view.getBye().addActionListener(e -> exitApp());
    }

    // Action for scanning a product
    1 usage  new *
    private void scanProduct() {
        int upc = scanner.generateUPC(); // Generate UPC code from Scanner
        Product product = model.getProduct(upc); // Fetch product details from CashRegister
        if (product != null) {
            // Update the text field to display UPC, Product name, and Price
            view.getFirstnameTextfield().setText("UPC: " + upc + " | Product: " + product.getName() + " | Price: $" + product.getPrice());
        } else {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Product not found.",  title: "Error", JOptionPane.ERROR_MESSAGE);
        }
    }


    1 usage  new *
    private void exitApp() {
        System.exit( status: 0); // Exit the application
    }
}
```

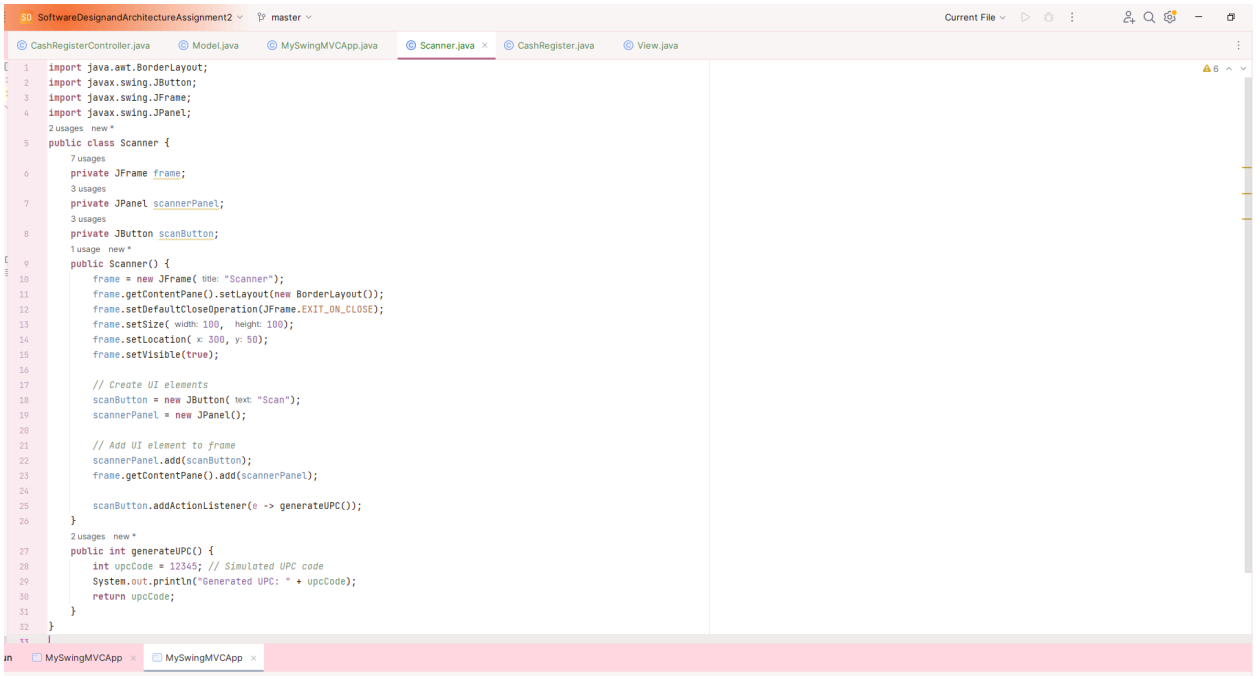
SoftwareDesignandArchitectureAssignment2 · master · Current File

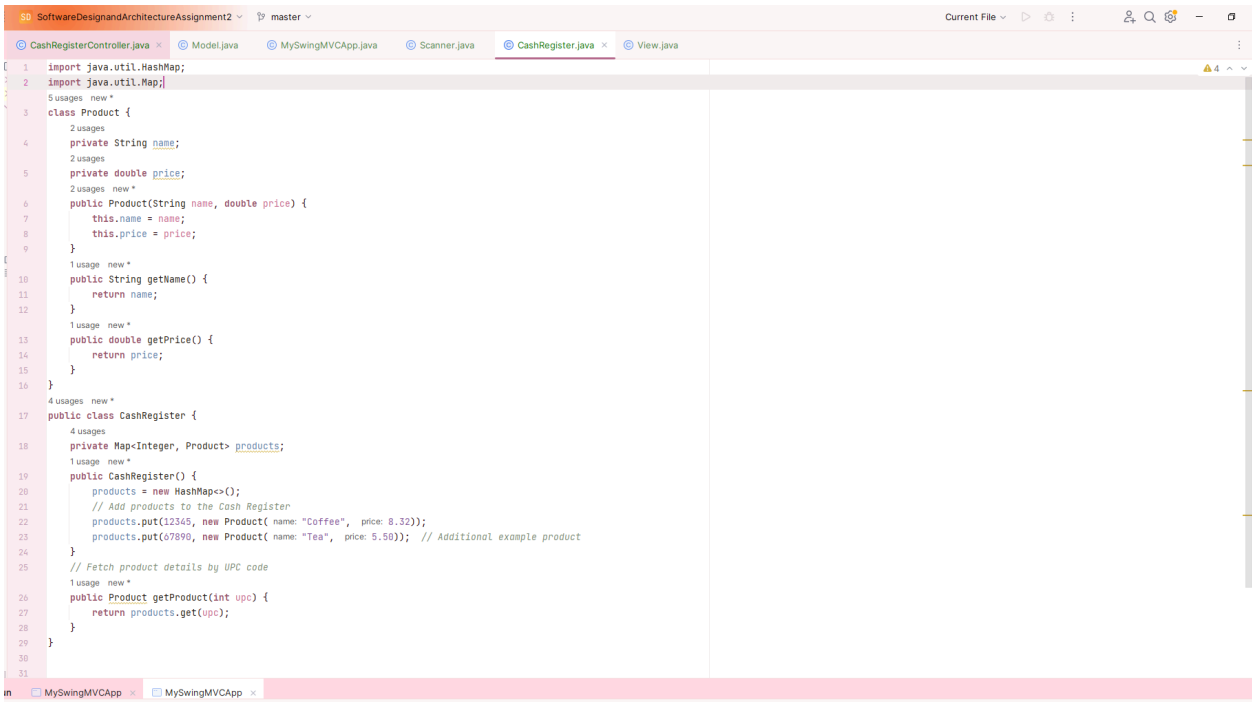
CashRegisterController.java | Model.java | MySwingMVCApp.java | Scanner.java | CashRegister.java | View.java

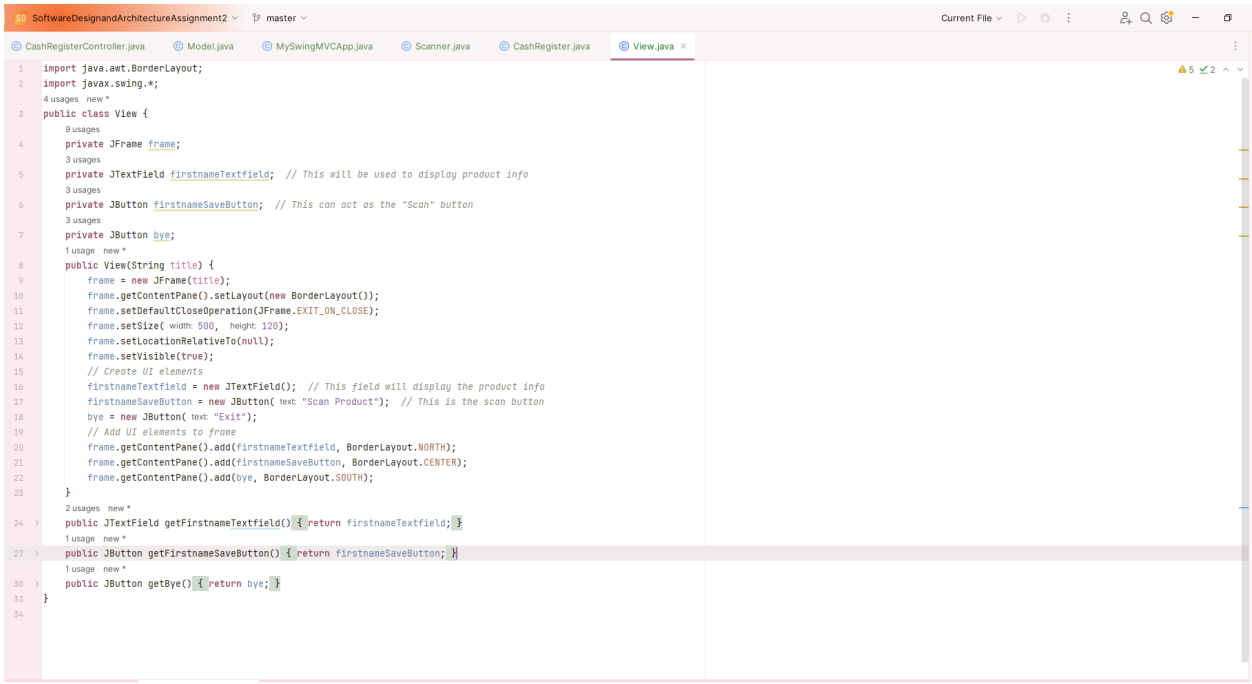
```
no usages  new *
public class Model {
    3 usages
    private String firstname;
    3 usages
    private String lastname;

    no usages  new *
    public Model(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    no usages  new *
    public String getFirstname() { return firstname; }

    no usages  new *
    public void setFirstname(String firstname) { this.firstname = firstname; }

    no usages  new *
    public String getLastname() { return lastname; }

    no usages  new *
    public void setLastname(String lastname) { this.lastname = lastname; }
}
```


SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
no usages new *
public class Model {
3 usages
private String firstname;
3 usages
private String lastname;
no usages new *
public Model(String firstname, String lastname) {
this.firstname = firstname;
this.lastname = lastname;
}
no usages new *
public String getFirstname() { return firstname; }
no usages new *
public void setFirstname(String firstname) { this.firstname = firstname; }
no usages new *
public String getLastname() { return lastname; }
no usages new *
public void setLastname(String lastname) { this.lastname = lastname; }
}


SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
new *
public class MySwingMVCApp {
new *
public static void main(String[] args) {
// Initialize Model (Cash Register), View, and Controller
CashRegister model = new CashRegister();
View view = new View(title: "Cash Register MVC");
CashRegisterController controller = new CashRegisterController(model, view);
controller.initController(); // Initialize controller
}
}


SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
2 usages new *
public class Scanner {
7 usages
private JFrame frame;
3 usages
private JPanel scannerPanel;
3 usages
private JButton scanButton;
1 usage new *
public Scanner() {
frame = new JFrame(title: "Scanner");
frame.getContentPane().setLayout(new BorderLayout());
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(width: 100, height: 100);
frame.setLocation(x: 300, y: 50);
frame.setVisible(true);
// Create UI elements
scanButton = new JButton(text: "Scan");
scannerPanel = new JPanel();
// Add UI element to frame
scannerPanel.add(scanButton);
frame.getContentPane().add(scannerPanel);
scanButton.addActionListener(e -> generateUPC());
}
2 usages new *
public int generateUPC() {
int upcCode = 12345; // Simulated UPC code
System.out.println("Generated UPC: " + upcCode);
return upcCode;
}
}
MySwingMVCApp
MySwingMVCApp


SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
import java.util.HashMap;
import java.util.Map;
5 usages new *
class Product {
2 usages
private String name;
2 usages
private double price;
2 usages new *
public Product(String name, double price) {
this.name = name;
this.price = price;
}
1 usage new *
public String getName() {
return name;
}
1 usage new *
public double getPrice() {
return price;
}
}
4 usages new *
public class CashRegister {
4 usages
private Map<Integer, Product> products;
1 usage new *
public CashRegister() {
products = new HashMap<>();
// Add products to the Cash Register
products.put(12345, new Product(name: "Coffee", price: 8.32));
products.put(67890, new Product(name: "Tea", price: 5.50)); // Additional example product
}
// Fetch product details by UPC code
1 usage new *
public Product getProduct(int upc) {
return products.get(upc);
}
}
MySwingMVCApp
MySwingMVCApp


SoftwareDesignandArchitectureAssignment2
master
Current File
CashRegisterController.java
Model.java
MySwingMVCApp.java
Scanner.java
CashRegister.java
View.java
5 2
import java.awt.BorderLayout;
import javax.swing.*;
4 usages new *
public class View {
9 usages
private JFrame frame;
3 usages
private JTextField firstnameTextfield; // This will be used to display product info
3 usages
private JButton firstnameSaveButton; // This can act as the "Scan" button
3 usages
private JButton bye;
1 usage new *
public View(String title) {
frame = new JFrame(title);
frame.getContentPane().setLayout(new BorderLayout());
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(width: 500, height: 120);
frame.setLocationRelativeTo(null);
frame.setVisible(true);
// Create UI elements
firstnameTextfield = new JTextField(); // This field will display the product info
firstnameSaveButton = new JButton(text: "Scan Product"); // This is the scan button
bye = new JButton(text: "Exit");
// Add UI elements to frame
frame.getContentPane().add(firstnameTextfield, BorderLayout.NORTH);
frame.getContentPane().add(firstnameSaveButton, BorderLayout.CENTER);
frame.getContentPane().add(bye, BorderLayout.SOUTH);
}
2 usages new *
public JTextField getFirstnameTextfield() { return firstnameTextfield; }
1 usage new *
public JButton getFirstnameSaveButton() { return firstnameSaveButton; }
1 usage new *
public JButton getBye() { return bye; }
}