

RV COLLEGE OF ENGINEERING®
BENGALURU – 560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



“FOOD ORDERING APPLICATION”

MINI-PROJECT REPORT
OBJECT ORIENTED PROGRAMMING USING JAVA (18CS45)
IV SEMESTER

2020-21

Submitted by

DARSHAN J - 1RV19CS042
HARIKIRAN G - 1RV19CS055

Under the Guidance of

Dr. Azra Nasreen

**Department of CSE,
RVCE,Bengaluru -
560059**

Dr. Pratiba D

**Department of CSE,
RVCE,Bengaluru -
560059**

Dr. Poonam Ghuli

**Department of CSE,
RVCE,Bengaluru -
560059**

RV COLLEGE OF ENGINEERING[®], BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

Certified that the **Mini**-project work titled “FOOD ORDERING APPLICATION” has been carried out by **DARSHAN J (USN - 1RV19CS042) and HARIKIRAN G (USN - 1RV19CS055)**, bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Assessment of Course: OBJECT ORIENTED PROGRAMMING USING JAVA (18SC45) – Open-Ended Experiments** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report.

Faculty Incharge
Department of CSE,
RVCE., Bengaluru –59

Head of Department
Department of CSE,
RVCE, Bengaluru–59

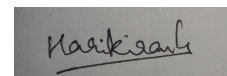
RV COLLEGE OF ENGINEERING[®], BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **DARSHAN J (USN - 1RV19CS042)** and **HARIKIRAN G (USN - 1RV19CS055)** the students of 4th Semester B.E., Department of Computer Science and Engineering, RV College of Engineering, Bengaluru hereby declare that the Mini-Project titled “**FOOD ORDERING APPLICATION**” has been carried out by us and submitted in partial fulfillment for the **Assessment of Course: OBJECT ORIENTED PROGRAMMING USING JAVA (16CS44) - Open-Ended Experiment** during the year 2020-2021.

Place: Bengaluru



Date: 10/08/2021

DARSHAN J

HARIKIRAN G

Signature

Contents

TITLE	Page No
1. Introduction	5
1.1 Object-Oriented Concepts	5
1.1.1 Traditional vs Object-Oriented Approach	5
1.1.2 OOA, OOD & OOP and their Relationship	5
1.1.3 Features of OOP approach	7
1.2 Overview of Java Programming Language	9
1.2.1 Features of Java	10
1.2.2 Inheritance	10
1.2.3 Interfaces & Packages	11
1.2.4 Exception Handling	12
1.2.5 Multithreaded Programming	14
1.2.6 Lambda Expressions	14
1.2.7 Regular Expressions	15
1.2.8 Strings	16
1.2.9 Collection Framework	17
1.2.10 JavaFX framework	18
1.3 Proposed System	19
1.3.1 Objectives	19
1.3.2 Methodology	20
1.3.3 Scope	20
2. Requirement Specifications	21
2.1 Hardware Requirements	21
2.2 Software Requirements	21
3. System Design and Implementation	22
3.1 Modular Description/ Pseudo-code	22
3.2 Class Diagrams	27
4. Results and Snapshots	29
5. Conclusion	37
6. References	37

1. INTRODUCTION

1.1. OBJECT-ORIENTED CONCEPTS

1.1.1. TRADITIONAL vs OBJECT-ORIENTED APPROACH

- The traditional approach to developing software systems that used procedural programming, could not be used to develop software because of some drawbacks that affected the efficiency and maintainability of the software.
- Object oriented approach to software engineering is used to cover up the drawbacks and develop software projects that use object oriented programming.
- In procedural programming, functions are described and called to perform the specific tasks, wherein the data is not encapsulated with the functions. This major problem comes up with the traditional approach where data is global and not encapsulated within any model object.
- Object oriented programming/paradigm was developed to solve or cover up the problem where all the components of the system as a real entity having attributes and functions linked with it. With this object-oriented approach, a blueprint or prototype of any entity can be described and is called a class, and various objects can be created from this.
- Object-oriented, programming solved some other problems related with traditional procedural programming such as security, effectiveness, data hiding, abstraction, inheritance etc. Object oriented programming also promotes communication within objects through the means of message passing.
- Considering all the points, it is not always that an Object-oriented approach will be easier in terms of difficulty level than the traditional approach. For traditional approaches, procedural programming depends on the size of the software programs, whereas difficulty levels of the object-oriented approach depend on the experience of the development team and the complexity of the programs.

To ensure a highly efficient and scalable software, an object-oriented approach was a necessary development over traditional approaches.

1.1.2. OOA, OOD & OOP and their Relationship

OBJECT-ORIENTED ANALYSIS

Object-Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects.

The main difference between object-oriented analysis and other forms of analysis is that requirements are organized around objects in an object-oriented approach, which integrates both data and functions. They are modeled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects - functions and data - are considered separately.

Grady Booch has defined OOA as “Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain”.

The primary tasks in object-oriented analysis (OOA) are –

- Identifying objects
- Organizing the objects by creating an object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

OBJECT-ORIENTED DESIGN

Object-Oriented Design (OOD) involves the implementation of the conceptual model produced during object-oriented analysis. In OOD, concepts in the analysis model, which are technology-independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

The implementation details generally include –

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,
- Implementation of control, and
- Implementation of associations.

Grady Booch has defined object-oriented design as “a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design”.

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming (OOP) is a programming paradigm based upon objects (having both data and methods) that aims to incorporate the advantages of modularity and reusability. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

The important features of object-oriented programming are –

- The bottom-up approach in program design
- Programs organized around objects, grouped in classes
- Focus on data with methods to operate upon object's data
- Interaction between objects through functions
- Reusability of design through the creation of new classes by adding features to existing classes

Some examples of object-oriented programming languages are C++, Java, Smalltalk, Delphi, C#, Perl, Python, Ruby, and PHP.

Grady Booch has defined object-oriented programming as “a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships”.

1.1.3 FEATURES OF OOP

Features of OOPs:

1. Classes
2. Objects
3. Data Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism

What is Class?

A class represents a real-world entity that acts as a blueprint for all the objects.

We can create as many objects as we need using Class.

What is an Object?

An object-Oriented Programming system(OOPS) is designed based on the concept of “Object”. It contains both variables (used for holding the data) and methods(used for defining the behaviors).

We can create any number of objects using this class and all those objects will get the same fields and behavior.

```
Student s1 = new Student();
```

Now we have created 3 objects s1,s2, and s3 for the same class “ Student ”.We can create as many objects as required in the same way.

We can set the value for each field of an object as below,

```
s1.id=123;
```

```
s2.age=18;  
s3.course="computers";
```

What is Abstraction?

Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user.

For example, when you log in to your bank account online, you enter your user_id and password and press login, what happens when you press login, how the input data is sent to the server, how it gets verified is all abstracted away from you.

We can achieve “ abstraction ” in Java using 2 ways

- Abstract class
- Interface

1. Abstract Class

Abstract class in Java can be created using the “ abstract ” keyword.

If we make any class abstract then it can’t be instantiated which means we are not able to create the object of the abstract class.

Inside Abstract class, we can declare abstract methods as well as concrete methods.

So using abstract class, we can achieve 0 to 100 % abstraction.

Anyone who needs to access this functionality has to call the method using the Phone object pointing to its subclass.

2. Interface

The interface is used to achieve pure or complete abstraction.

We will have all the methods declared inside Interface as abstract only.

So, we call interface 100% abstraction.

Now, these functionalities like changing gear and applying brakes are abstracted using this interface.

What is Encapsulation?

Encapsulation is the process of binding object state(fields) and behaviors(methods) together in a single entity called “Class”.

Since it wraps both fields and methods in a class, it will be secured from the outside access.

We can restrict the access to the members of a class using access modifiers such as private, protected, and public keywords.

When we create a class in Java, it means we are doing encapsulation.

Encapsulation helps us to achieve the re-usability of code without compromising the security.

What is the benefit of encapsulation in java programming?

Well, at some point in time, if you want to change the implementation details of the class EmployeeCount, you can freely do so without affecting the classes that are using it. For more information learn,

What is Inheritance?

One class inherits or acquires the properties of another class.

Inheritance provides the idea of reusability of code and each subclass defines only those features that are unique to it, the rest of the features can be inherited from the parent class.

Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.

It allows us to reuse code, it improves reusability in your java application.

The parent class is called the base class or superclass. The child class that extends the base class is called the derived class or subclass or child class.

To inherit a class we use extends keyword. Here class A is child class and class B is parent class.

```
class A extends B
{ }
```

Types Of Inheritance:

1. Single Inheritance: refers to a child and parent class relationship where a class extends another class.
2. Multilevel inheritance: a child and parent class relationship where a class extends the child class. For example, class A extends class B and class B extends class C.
3. Hierarchical inheritance: where more than one class extends the same class. For example, class B extends class A and class C extends class A.

What is Polymorphism?

It is the concept where an object behaves differently in different situations.

Since the object takes multiple forms, it is called Polymorphism.

In java, we can achieve it using method overloading and method overriding.

There are 2 types of Polymorphism available in Java,

- Method overloading
In this case, which method to call will be decided at the compile time itself based on the number or type of the parameters. Static/Compile Time polymorphism is an example of method overloading.
- Method overriding
In this case, which method to call will be decided at the run time based on what object is to be to, pointed by the reference variable.

1.2 OVERVIEW OF JAVA

1.2.1 FEATURES OF JAVA

1. Simple: Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, Explicit memory allocation.

2. Robust: Java language is robust that means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler can detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.
3. Secure: In java, we don't have pointers, and so we cannot access out-of-bound arrays i.e it shows `ArrayIndexOutOfBoundsException` if we try to do so. That's why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java.
4. Distributed: We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected through an internet connection.
5. Multithreading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.
6. Portable: As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

IMPLEMENTATION OF JAVA CONCEPTS IN OUR PROJECT

1.2.2 INHERITANCE

Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and adds its own, unique elements.

For example,

```
public class Cuisine extends Food{
```

Here, class Cuisine inherits all the characteristics of the Food class. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

1.2.3 INTERFACE AND PACKAGES

An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface, not method body. It cannot be instantiated just like abstract class and interface is not extended by a class; it is implemented by a class. An interface can extend multiple interfaces. The interface keyword is used to declare an interface.

Syntax: `public interface NameOfInterface`

Interfaces have the following properties: –

- An interface is implicitly abstract.
- We do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Implementing Interfaces: A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extended portion of the declaration.

In our project, we have implemented two interfaces, `MyListener`, and `validation`. Both interfaces are functional interfaces and their implementation is defined using a lambda expression.

```
package Main;

import Model.Food;

public interface MyListener {
    void onClickListener(Food food);
}
```

```
package Model;

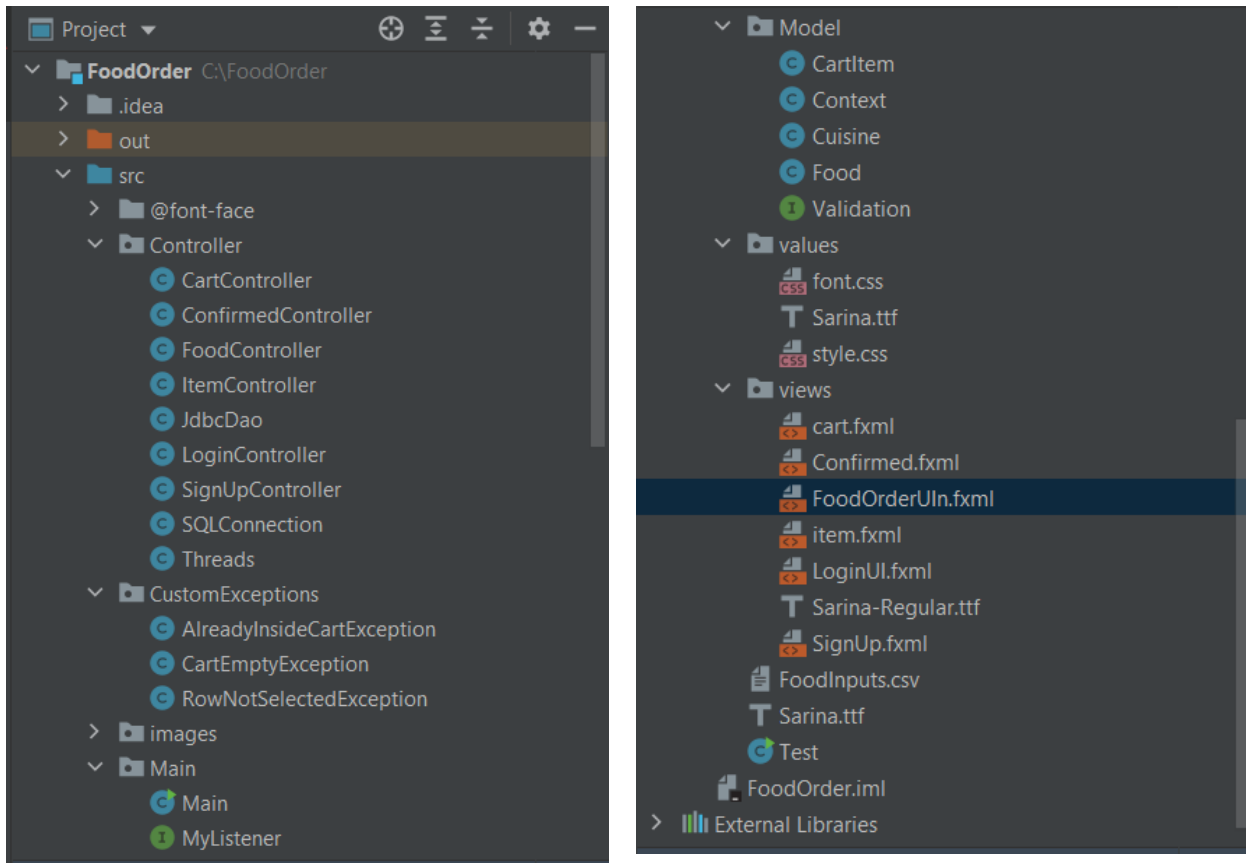
public interface Validation {
    boolean validate();
}
```

Packages:

A java package is a group of similar types of classes, interfaces, and sub-packages. Package in java can be categorized in two forms, built-in package, and user-defined package. There are many built-in packages such as `java`, `lang`, `awt`, `javax`, `swing`, `net`, `io`, `util`, `SQL`, etc. Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collisions.



1.2.4 EXCEPTION HANDLING

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself or pass it on. Either way, at some point, the exception is caught and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method.

We have implemented three user-defined exceptions. They include:

- AlreadyInsideCartException

```
package CustomExceptions;

public class AlreadyInsideCartException extends RuntimeException{
    public String toString(){
        return "!! Already Inside Cart !!";
    }
}
```

- CartEmptyException

```
package CustomExceptions;

public class CartEmptyException extends RuntimeException{
    public String toString(){
        | return "!! Cart is Empty !!";
    }
}
```

- RowNotSelectedException

```
package CustomExceptions;

public class RowNotSelectedException extends RuntimeException{
    public String toString(){
        return "!! Please select row to apply specific action !!";
    }
}
```

The following code snippet gives a basic idea of how user-defined exceptions work. We need to specify what exception a method might throw in the function definition using the **throws** keyword. . Next, the program statements that we want to monitor for exceptions are contained within a **try** block. If an exception occurs within the try block, it is thrown. The code can catch this exception (using **catch**) and handle it in some rational manner.

System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword **throw**.

```
public void handle(ActionEvent e) throws AlreadyInsideCartException
{
    try {
        if (FoodName.contains(chosenFood.getName())) {
            throw new AlreadyInsideCartException();
        }
    }
    catch (AlreadyInsideCartException e1)
    {
        AlertLabel.setVisible(true);
        AlertLabel.setText(e1.toString());
        System.out.println("Exception caught and handled : " + e1);
        return;
    }
}
```

AlreadyInsideCartException will throw an exception when the selected food item is already inside the cart and perform the necessary exception handling procedure as specified in the catch block. Similarly, CartEmptyException throws an exception when the cart is empty and RowNotSelectedException throws an exception when no row is selected to perform operations in the cart table.

1.2.5 MULTITHREADED PROGRAMMING

Unlike many other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

Process-based multitasking vs Thread-based multitasking - Process-based multitasking is the feature that allows your computer to run two or more programs concurrently. For example, process-based multitasking enables you to run the Java compiler at the same time that you are using a text editor. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

In a thread-based multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously. For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads. Thus, process-based multitasking deals with the “big picture,” and thread-based multitasking handles the details.

```
Threads thread1 = new Threads(SouthCuisine.getStarts(),NorthCuisine.getStarts(), searchText.getText().toString());
Threads thread2 = new Threads( SouthCuisine.getMainCourse(),NorthCuisine.getMainCourse(), searchText.getText().toString());
Threads thread3 = new Threads(SouthCuisine.getSweets(), NorthCuisine.getSweets(), searchText.getText().toString());
Threads thread4 = new Threads( SouthCuisine.getDrinks(),NorthCuisine.getDrinks(), searchText.getText().toString());
thread1.start();
thread2.start();
thread3.start();
thread4.start();
thread1.join();
thread2.join();
thread3.join();
thread4.join();
```

In the above code snippet, we are creating four threads. Each thread is assigned the task to search for a given input string in 4 different lists each. This improves performance drastically because all 4 threads can search simultaneously. Each thread is started by calling its run method using `.start()` and `.join()` is used to wait for a thread to terminate.

1.2.6 LAMBDA EXPRESSIONS

Lambda expressions are introduced in Java 8 and are touted to be the biggest feature of Java 8. Lambda expression facilitates functional programming and simplifies the development a lot.

A lambda expression can implement a functional interface by defining an anonymous function that can be passed as an argument to some method.

- Enables functional programming: All new JVM based languages take advantage of the functional paradigm in their applications, but programmers were forced to work with Object-Oriented Programming (OOPS) till lambda expressions came. Hence lambda expressions enable us to write functional code.
- Readable and concise code: People have started using lambda expressions and reported that it can help to remove a huge number of lines from their code.

- Easy-to-Use APIs and Libraries: An API designed using lambda expressions can be easier to use and support other APIs.
- Enables support for parallel processing: A lambda expression can also enable us to write parallel processing because every processor is a multi-core processor nowadays.

Following are the important characteristics of a lambda expression.

- Optional type declaration – No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
- Optional parentheses around parameter – No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- Optional curly braces – No need to use curly braces in the expression body if the body contains a single statement.
- Optional return keyword – The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that the expression returns a value.

For example,

```
Validation EmailValidate = () -> {
```

This simplifies the writing of code, since we need not have to create a separate function for this and helps in reducing the number of lines of code.

1.2.7 REGULAR EXPRESSIONS

A regular expression (regex) defines a search pattern for strings. The search pattern can be anything from a simple character, a fixed string, or a complex expression containing special characters describing the pattern. A regex can be used to search, edit and manipulate text, this process is called: The regular expression is applied to the text/string.

The regex is applied to the text from left to right. Once a source character has been used in a match, it cannot be reused. For example, the regex aba will match ababababa only two times (aba_aba__).

We have implemented regex for two tasks:

1. Email Validation:

```
Validation EmailValidate = () -> {  
    String regex = "[A-Za-z0-9+_.-]+@(.+)$";  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(EmailField.getText());  
    if(!matcher.matches())  
    {  
        EMError.setVisible(true);  
        EMError.setText("Invalid Email");  
    }  
    return matcher.matches();  
};
```

2. Phone Number Validation:

```
Validation PhNumValidate = () -> {  
    String regex = "[0-9]{10}";  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(PhNumField.getText());  
    if(!matcher.matches()) {  
        PHNumError.setVisible(true);  
        PHNumError.setText("Invalid Phone\nNumber");  
    }  
    return matcher.matches();  
};
```

1.2.8 STRINGS

As is the case in most other programming languages, in Java a string is a sequence of characters. But, unlike many other languages that implement strings as character arrays, Java implements strings as objects of type `String`. Implementing strings as built-in objects allows Java to provide a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string. Also, `String` objects can be constructed in several ways, making it easy to obtain a string when needed.

When you create a `String` object, you are creating a string that cannot be changed. That is, once a `String` object has been created, you cannot change the characters that comprise that string. At first, this may seem to be a serious restriction. However, such is not the case. You can still perform all types of string operations. The difference is that each time you need an altered version of an existing string, a new `String` object is created that contains the modifications. The original string is left unchanged. This approach is used because fixed, immutable strings can be implemented more efficiently than changeable ones.

In our implementation, we have a CSV file containing a table of food items, their prices, several controllers, and under which cuisine it falls. We are extracting these details as strings and storing them in lists.

```
try {
    File file = new File( pathname: "C:\\FoodOrder\\src\\FoodInputs.csv");
    FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr);
    String line = "";
    String[] tempArr;
    while ((line = br.readLine()) != null) {
        tempArr = line.split(delimiter);
        Food food = new Food();
        food.setName(tempArr[2]);
        food.setImgSrc(tempArr[3]);

        food.setPrice(Double.parseDouble(tempArr[4]));
        food.setColor(tempArr[5]);
    }
}
```

After reading these values, we store the food object in the respective cuisine list, for instance, SouthCuisine.Starters, NorthCuisine.Starters, etc.

Another case of string handling is in the search implementation as mentioned earlier in the threads segment.

```
public void LinearSearch()
{
    int size1 = this.list1.size();
    int size2 = this.list2.size();
    for(int i=0 ; i<size1 ; i++) {
        if(this.list1.get(i).getName().equalsIgnoreCase(toSearch)) {
            System.out.println(list1.get(i).getName() + "Index : " + i);
            this.result = i;
            return;
        }
    }
}
```

This code snippet shown above is for one list. We are comparing the list of food items' names with the given **search** string using the **.equalsIgnoreCase()** function. Whenever a match is found, we are printing the index of that match and setting the result to the matched index and will be later used for setting the chosenFoodCard to the matched food index.

1.2.9 COLLECTION FRAMEWORK

The Java Collections Framework standardizes how groups of objects are handled by your programs. Collections were not part of the original Java release but were added by J2SE 1.2.

The Collections Framework was designed to meet several goals. First, the framework had to be high-performance. The implementations for the fundamental collections (dynamic arrays, linked lists, trees, and hash tables) are highly efficient. You seldom, if ever, need to code one of these “data engines” manually. Second, the framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability. Third, extending and/or adapting a collection had to be easy. Toward this end, the entire Collections Framework is built upon a set of standard interfaces. Several standard implementations (such as LinkedList, HashSet, and TreeSet) of these interfaces are provided that we may use as-is.

Implementations:

1. ObservableArrayList

```
ObservableList<String> QuantityList = FXCollections.observableArrayList( ...es: "1", "2", "3", "4", "5");
```

We have used this to store the numbers 1 to 5 in the select quantity dropdown box on the FoodOrderUI page.

2. ObservableArrayList

```
ObservableList<CartItem> cartList = FXCollections.observableArrayList();
```

We have used this to store the list of items that will be displayed in the cart table.

3. ArrayList

```
private final List<Food> Starters = new ArrayList<>();  
private final List<Food> MainCourse = new ArrayList<>();  
private final List<Food> Sweets = new ArrayList<>();  
private final List<Food> Drinks = new ArrayList<>();
```

Four ArrayLists to store the food items under the respective lists.

1.2.10 JAVAFX FRAMEWORK

JavaFX is an open-source Java-based framework for developing rich client applications. It is comparable to other frameworks on the market such as Adobe Flex and Microsoft Silverlight. JavaFX is also seen as the successor of Swing in the arena of graphical user interface (GUI) development technology in the Java platform. The JavaFX library is available as a public Java application programming interface (API).

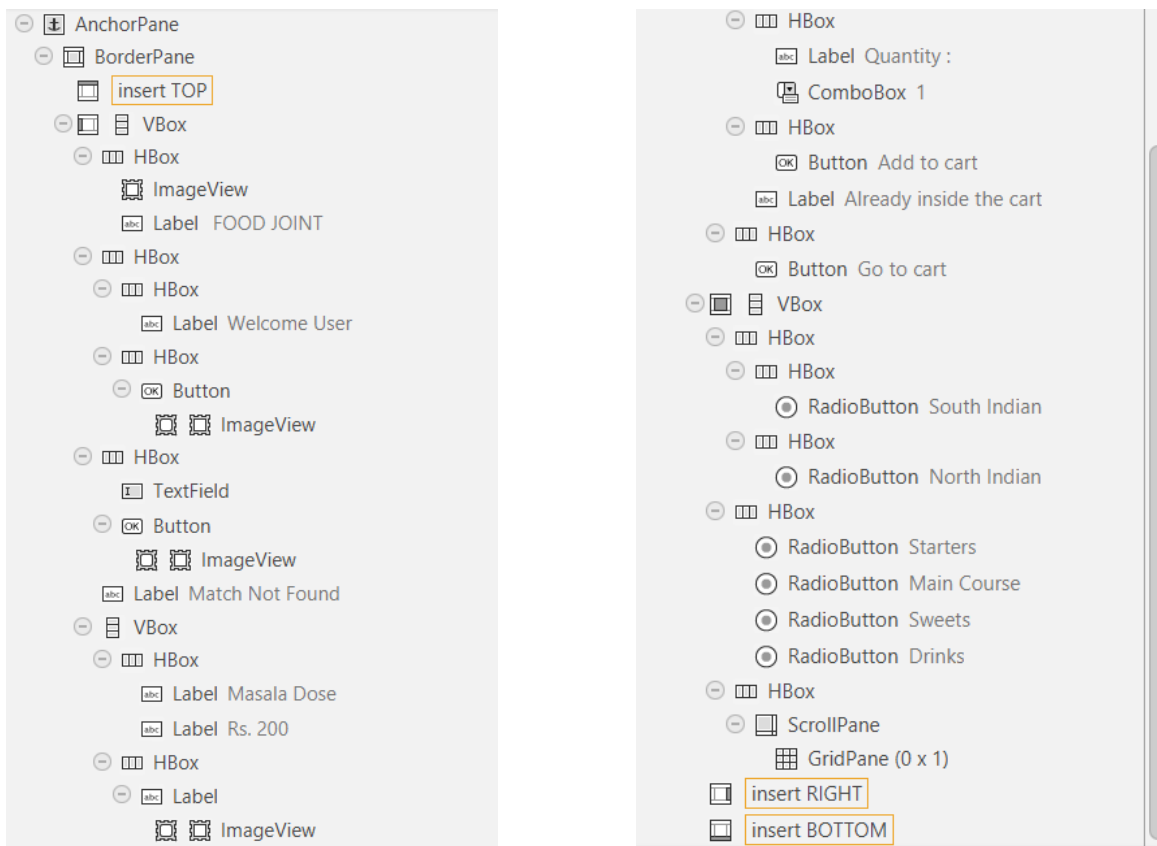
JavaFX contains several features that make it a preferred choice for developing rich client applications:

- JavaFX is written in Java, which enables you to take advantage of all Java features such as multithreading, generics, and lambda expressions. You can use any Java editor of your choice, such as NetBeans, to the author, compile, run, debug, and package your JavaFX application.
- JavaFX supports data binding through its libraries.
- JavaFX code can be written using any Java virtual machine (JVM)-supported scripting languages such as Visage, Groovy, and Scala.
- JavaFX offers two ways to build a user interface (UI): using Java code and using FXML. FXML is an XML-based scriptable markup language to define a UI declaratively. Oracle provides a tool called Scene Builder, which is a visual editor for FXML.

- JavaFX provides a rich set of multimedia support such as playing back audios and videos. It takes advantage of available codecs on the platform.
- JavaFX lets you embed web content in the application.
- JavaFX provides out-of-the-box support for applying effects and animations, which are important for developing gaming applications. You can achieve sophisticated animations by writing a few lines of code.

JavaFX elements in our project: We have used almost all basic JavaFX elements like buttons, labels, ImageView, Combobox, Togglebuttons, Actionevents. In addition to this we have used GridPane and ScrollPane for items view, TableView for cart items display,

For example, this is the hierarchical layout of nodes of one of our scenes: FoodOrderUI.fxml



1.3 PROPOSED SYSTEM

1.3.1 OBJECTIVES

We started our project keeping in mind the following objectives:

- To develop a user-friendly JavaFX GUI-based Food Ordering Application.
- To implement the principles and features of Object-Oriented Programming.
- To understand the working and controls of JavaFX.

1.3.2 METHODOLOGY

In our Application, there are 5 scenes/pages.

- Login Page.
- Registration Page.
- Main Page.
- Cart Page.
- Order Confirmation Page.

Every page is controlled by its respective Controller class. It has definitions of all the functionalities seen in the respective pages/scenes.

Login Page and Registration Page:

The landing page of our application is the Login page where the user can enter his username and password and can log in to the application. We have used TextFields to take input from the user and labels for prompts and buttons. MySQL has been used for database and login confirmation. In case if the user has not registered, there is an option for registration, by clicking the registration button the user is taken to the registration page. On the Registration page, one can enter a username, email id, phone number, and password. With the help of regex email id and username are validated. If all conditions are met registration is successful and the user can log in to the application using registered credentials. LoginController for Login Page and SignUpController for Registration page.

Main Page:

The main page layout is divided into two parts. One is a menu gallery and another is a details gallery. In the menu gallery, there are two groups of toggle buttons. From one group, the user can choose which cuisine he/she is interested in, in our case North Indian or South Indian. Another group contains sub-menu like Starters, Main Course, Desserts, Drinks. Upon selecting any combinations respective items are displayed below. All info about items is stored in a CSV file.

The details gallery, the one in the left part of the screen, contains the Application name, logo, Logged in username, logout button, Search layout, Chosen-item-details card, Go-to-cart button. In the chosen-item-details card, details like price serving quantity, Quantity comboBox, and Add-to-cart button are placed. Controller class is FoodController.

Cart Page and Order Confirmation page:

A list of items that are added to the cart using the Add-to-cart button is displayed on the cart Page using TableView. Users can increase the quantity or decrease the quantity or remove any items from the list. Users can go back to the Main Page with the help of a Back-to-home button. A label displays the total checkout amount. A confirm order button places the order. Upon clicking this button a new page is displayed saying theThe controller thank you message for placing an order with the username. CartController class and ConfirmedController for Confirmation Page.

For building the scenes for all the above pages we have used SceneBuilder. With the help of it, we can easily build any difficult UIs for applications. It converts the layout drawn or built into an FXML file. Using FXMLloader we can load it whenever and wherever we want.

2. REQUIREMENT SPECIFICATIONS

2.1 Hardware Requirements:

- Processor - Intel Pentium 4, Intel Centrino, Intel Xeon, or Intel Core Duo (or compatible) 1.8 GHz minimum (2.6 GHz Intel Pentium 4 or equivalent recommended) or above.
- OS - Windows XP with Service Pack 3 or Windows Vista Home Premium, Business, Ultimate, or Enterprise (certified for 32-bit editions) or Windows 7 Ultimate (certified for 32-bit editions) or above.
- Memory - 512 MB of RAM (2 GB recommended)
- Disk Space - 256MB of free disk space (1 GB recommended)

2.2 Software Requirements:

- IntelliJ IDEA IDE
- Scenebuilder
- JDK 16
- JavaFX Library

3. SYSTEM DESIGN AND IMPLEMENTATION

3.1 MODULAR DESCRIPTION

1) FoodController

```
FoodController

~ event : EventHandler<ActionEvent>
+ delimiter : String [readOnly]
+ chosenFood : Food
+ quantity : int
~ SouthCuisine : Cuisine
~ NorthCuisine : Cuisine
~ FoodName : List<String>
~ cartItemList : List<CartItem>
- s : String
- IsNorth : boolean
- IsSouth : boolean
- myListener : MyListener
~ QuantityList : ObservableList<String>
- Serve : Label
- grid : GridPane
- welcomeUser : Label
- Quantity : ComboBox<String>
- searchText : TextField
- AlertLabel : Label
- Drinks : RadioButton
- Sweets : RadioButton
- MainCourse : RadioButton
- Starters : RadioButton
- NorthIndian : RadioButton
- SouthIndian : RadioButton
- TgMainMenu : ToggleGroup
- TgCuisine : ToggleGroup
- AddToCart : Button
- FoodImage : ImageView
- FoodPricelabel : Label
- SearchError : Label
- FoodNameLabel : Label
- ChosenFoodCard : VBox

+ setChosenFood(food : Food) : void
+ initialize(url : URL, resourceBundle : ResourceBundle) : void
+ QuantitySelect(event : ActionEvent) : void
+ LoadItems(foodL : List<Food>) : void
+ goToCart(event : ActionEvent) : void
+ SelectionFunction(s : String) : void
+ search(event : ActionEvent) : void
+ LogOut(event : ActionEvent) : void
```

- setChosenFood(Food food): This function takes item Food as a parameter and sets the ChosenFoodCard on the left side of the main UI page. It sets the food name, image, price, and card color.
- initialize(): This will load all food items from the CSV file to the lists and also has the functionality to call LoadItems().
- QuantitySelect() - set item quantity from drop down list (comboBox).
- LoadItems(List<Food>) - This method loads corresponding food items to the grid pane in the right portion of the main UI page, on clicking the toggle buttons.
- search() - This method consists of the thread declaration, start and join functions for search functionality.
- goToCart() - This method is called when the go-to-cart button is pressed and the cart page is loaded onto the window.
- SelectionFuncton()- This method loads the submenu items on the display gallery. There are four submenus and respective items are displayed using the list implemented in cuisines class.
- LogOut() - This method is used to log out a user from his session. This is called

2) CartController

- initialize() - overridden initialize method. In this block we can add some elements which

<pre> CartController ~ FoodName : List<String> ~ cartItemList : List<CartItem> ~ cartList : ObservableList<CartItem> + cartItems : List<Food> - CartEmptyLabel : Label - SelectLabel : Label - totalAmountLabel : Label - foodAmount : TableColumn<CartItem, Double> - foodPrice : TableColumn<CartItem, Double> - foodQuantity : TableColumn<CartItem, Integer> - foodName : TableColumn<CartItem, String> - serialNo : TableColumn<CartItem, Integer> - cartTable : TableView<CartItem> - confirmOrder(event : ActionEvent) : void - DecreaseQuant() : void - IncreaseQuant() : void - deleteRowTable() : void + LoadItems(CartL : List<CartItem>) : void + goBacktoMain(event : ActionEvent) : void + initialize(url : URL, resourceBundle : ResourceBundle) : void </pre>	<p>are needed to be initialized when the screen is loaded.</p> <ul style="list-style-type: none"> • LoadItems(List<CartItem>) Given a list of cartItems, this will set the data (i.e., serialNo, foodName, foodQuantity, foodPrice, foodAmount) to the cart table. • deleteRowTable() - deletes a row in the cart table • DecreaseQuant() - reduce quantity of selected cart item in cart table by 1. • IncreaseQuant() - increase quantity of selected cart items in cart table by 1. • ConfirmOrder() - places the order and loads the confirmation screen. • gobacktoMain() - takes the user back to the main page.
<p>3) LoginController</p> <pre> LoginController ~ resultSet : ResultSet ~ preparedStatement : PreparedStatement ~ connection : Connection ~ passw : String ~ username : String ~ scene : Scene ~ dialogStage : Stage - InvalidError : Label - Password : PasswordField - NameField : TextField - LoginBP : BorderPane + initialize(url : URL, rb : ResourceBundle) : void + SignUp(event : ActionEvent) : void + loginAction(event : ActionEvent) : void + getUsername() : String + LoginController() </pre>	<ul style="list-style-type: none"> • initialize() - used to initialize some elements when the scene loads. • SignUp() - When Register Button is pressed this method is called. It loads the Registration Page to the screen. • LoginAction() - fetches the credentials and validates it using SQLconnection and queries. If the credentials are correct it will take the user to the Main Page. • getUsername() - used to return the name of the user who has logged in to greet him/her on the main page. • LoginController() - constructor used to get the SQL connection instance.
<p>4) SignUpController</p>	<ul style="list-style-type: none"> • GoBackToLogin() - loads the Login page when an associated button is pressed. • Showalert() - when registration is successful then an alert window pops up saying the same. • SignUp() - fetches the values from the TextFields, validates, it using regex and inserts it into the database.

<pre> SignUpController - SignUpBP : BorderPane - PHNumError : Label - SignUpBtn : Button - PWEError : Label - UNError : Label - EMEError : Label - PhNumField : TextField - EmailField : TextField - UserNameField : TextField - PasswordField : PasswordField + initialize(url : URL, rb : ResourceBundle) : void + GoBackToLogin() : void - showAlert(owner : Window, message : String) : void + SignUp(event : ActionEvent) : void </pre>	
<p>5) ItemController</p> <pre> ItemController - myListener : MyListener - food : Food - Img : ImageView - Pricelabel : Label - Namelabel : Label + setData(food : Food, myListener : MyListener) : void - click(mouseEvent : MouseEvent) : void </pre>	<ul style="list-style-type: none"> • In FoodController many objects are created of this class and set to each tile on the right side gallery of the application. This tile displays the name, image, and price of the item. • Each tile has a mylistener as an instance variable and lambda expression is passed through the setData() method when a user clicks on the tile particular lambda implementation is called, in this case, setChooseFood is called and details are displayed on the left display gallery.

Under the model package we have

- Food
- Cuisine
- Context
- CartItem
- Validation - Functional Interface.

1) Food Class

```
package Model;

public class Food {
    private String name;
    private String imgSrc;
    private double price;
    private String cardColor;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getImgSrc() { return imgSrc; }

    public void setImgSrc(String imgSrc) { this.imgSrc = imgSrc; }

    public double getPrice() { return price; }

    public void setPrice(double price) { this.price = price; }

    public String getColor() { return cardColor; }

    public void setColor(String color) { this.cardColor = color; }
}
```

Food

- cardColor : String
- price : double
- imgSrc : String
- name : String

+ setColor(color : String) : void
+ getColor() : String
+ setPrice(price : double) : void
+ getPrice() : double
+ setImgSrc(imgSrc : String) : void
+ getImgSrc() : String
+ setName(name : String) : void
+ getName() : String

2) Cuisine Class - This class extends Foodclass

```
package Model;

import java.util.ArrayList;
import java.util.List;

public class Cuisine extends Food{
    private final List<Food> Starters = new ArrayList<>();
    private final List<Food> MainCourse = new ArrayList<>();
    private final List<Food> Sweets = new ArrayList<>();
    private final List<Food> Drinks = new ArrayList<>();

    public void addDrinks(Food drinks) { Drinks.add(drinks); }

    public void addStarters(Food starter) { Starters.add(starter); }

    public void addSweets(Food sweets) { Sweets.add(sweets); }

    public void addMainCourse(Food mainCourse) { MainCourse.add(mainCourse); }

    public List<Food> getDrinks() { return Drinks; }

    public List<Food> getMainCourse() { return MainCourse; }

    public List<Food> getStarters() { return Starters; }

    public List<Food> getSweets() { return Sweets; }
}
```

Cuisine

- Drinks : List<Food> {readOnly}
- Sweets : List<Food> {readOnly}
- MainCourse : List<Food> {readOnly}
- Starters : List<Food> {readOnly}

+ addDrinks(drinks : Food) : void
+ addStarters(starter : Food) : void
+ addSweets(sweets : Food) : void
+ addMainCourse(mainCourse : Food) : void
+ getDrinks() : List<Food>
+ getMainCourse() : List<Food>
+ getStarters() : List<Food>
+ getSweets() : List<Food>

- 3) Context Class - The main use of this class is to maintain data between scene switching whenever new scene loads data from the ,previous scene will be lost, to overcome this Context class is built.

```
package Model;

import java.util.ArrayList;
import java.util.List;

public class Context {
    private final static Context instance = new Context();

    public static Context getInstance() { return instance; }

    private List<CartItem> cartItem = new ArrayList<>();
    List<String> FoodName = new ArrayList<>();

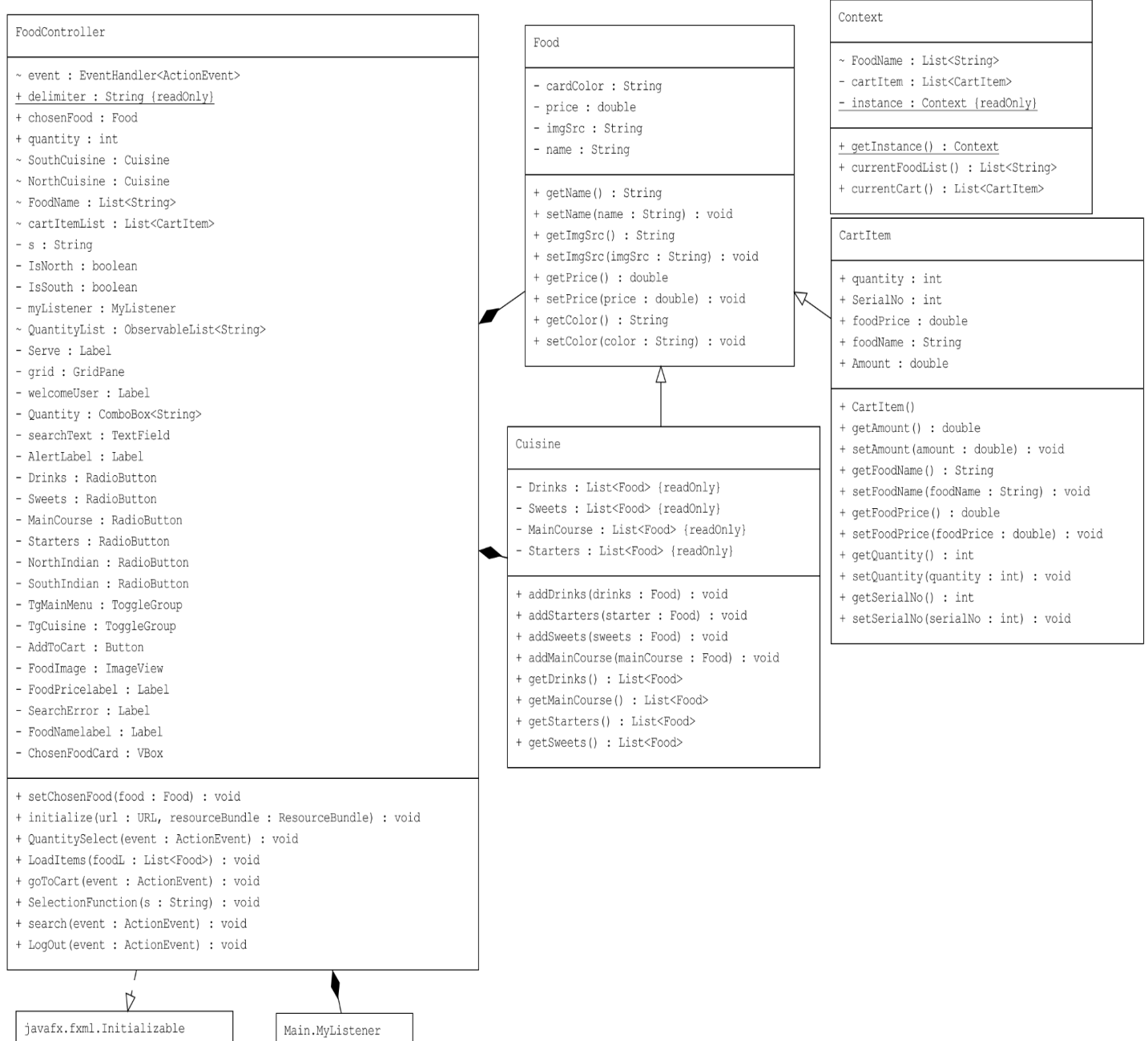
    public List<String> currentFoodList() {return FoodName;}
    public List<CartItem> currentCart() { return cartItem; }
}
```

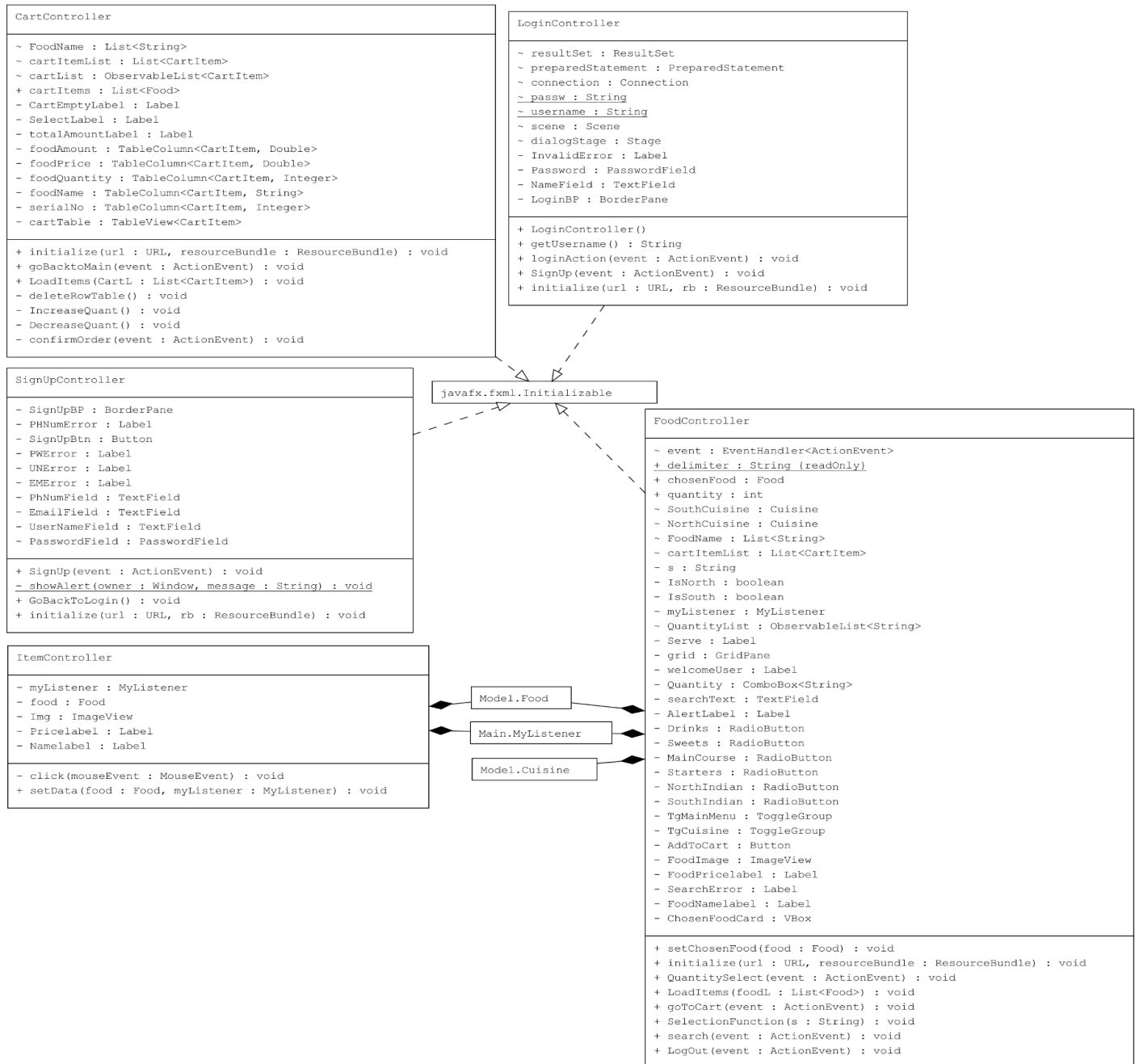
Context

~ FoodName : List<String>
- cartItem : List<CartItem>
- instance : Context {readOnly}

+ getInstance() : Context
+ currentFoodList() : List<String>
+ currentCart() : List<CartItem>

3.2 CLASS DIAGRAMS





4. RESULTS and SNAPSHOTS

The image displays two screenshots of a web application titled "Food Joint". Both screenshots feature a dark green sidebar on the left with the "FOOD JOINT" logo (a fork and knife icon) and a stylized illustration of a plate with a fried egg, vegetables, and fruit. The main content area is a lighter green.

Top Screenshot: SIGN IN


- Header: "SIGN IN" with a lock icon.
- Form fields: "USERNAME" (containing "admin1") and "PASSWORD" (containing "*****").
- Message: "Invalid Credentials" in red text.
- Buttons: "LOGIN" and "REGISTER NOW" (with the text "Not yet Registered?" preceding it).

Bottom Screenshot: SIGN UP

- Header: "SIGN UP" with a user icon.
- Form fields: "USERNAME", "EMAIL" (containing "Ex: abcd@gmail.com"), "PH NUMBER" (containing "10 Digits"), and "PASSWORD".
- Buttons: "SIGN UP" and "Go Back To Login Page".

Food Joint

FOOD JOINT

SIGN UP 

USERNAME Field is empty

EMAIL Field is Empty

PH NUMBER Field is Empty


PASSWORD Field is Empty

SIGN UP

Go Back To Login Page

Food Joint

FOOD JOINT

SIGN UP 

USERNAME

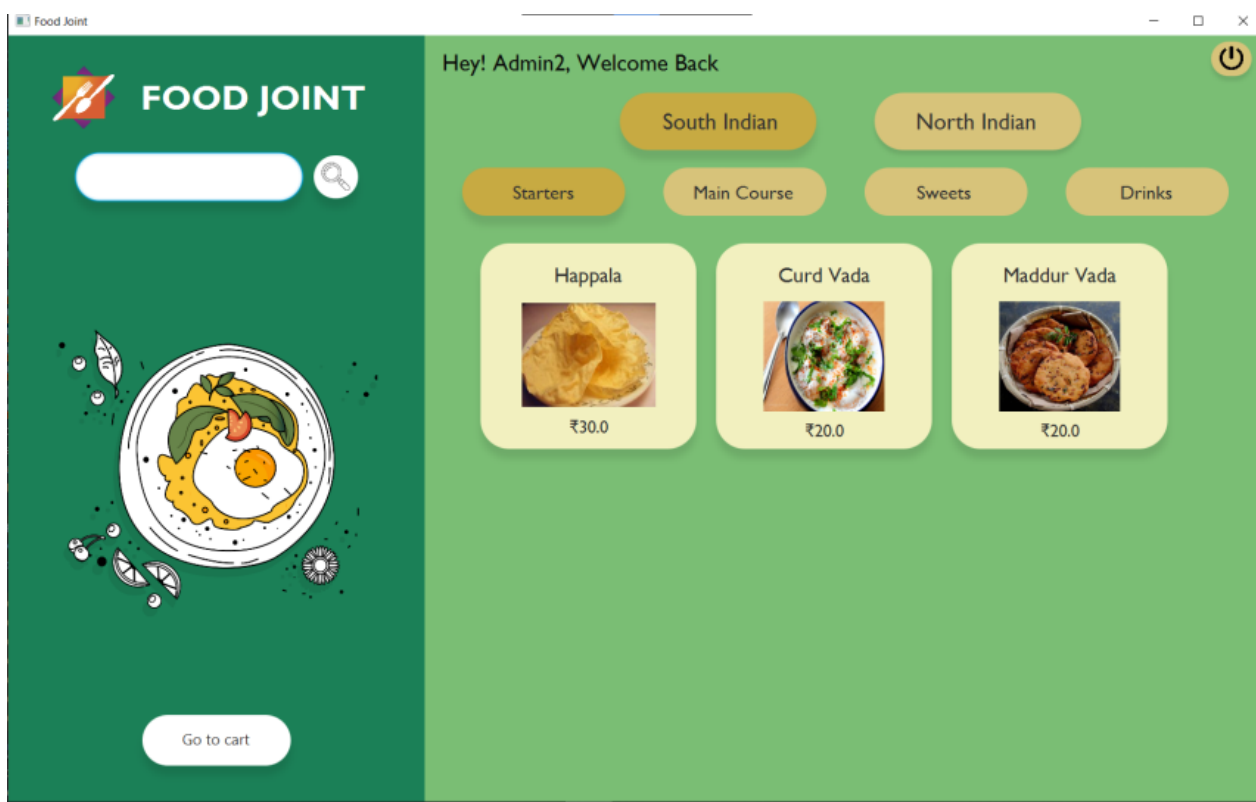
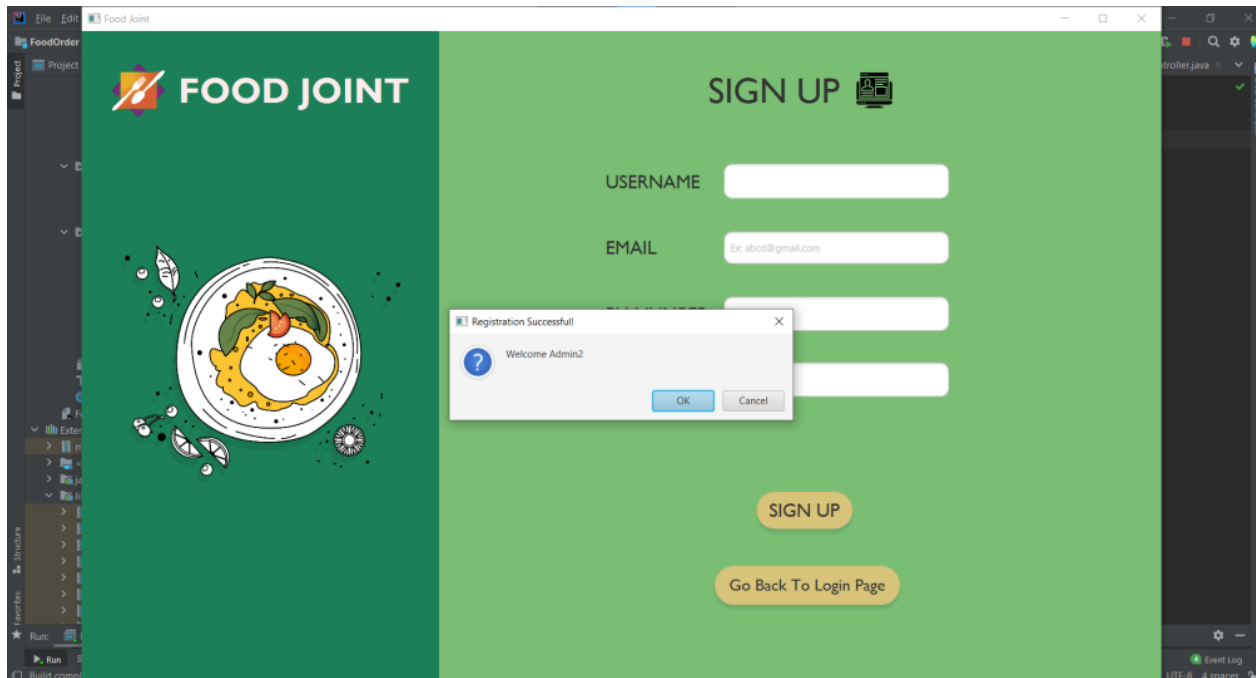
EMAIL Invalid Email

PH NUMBER Invalid Phone

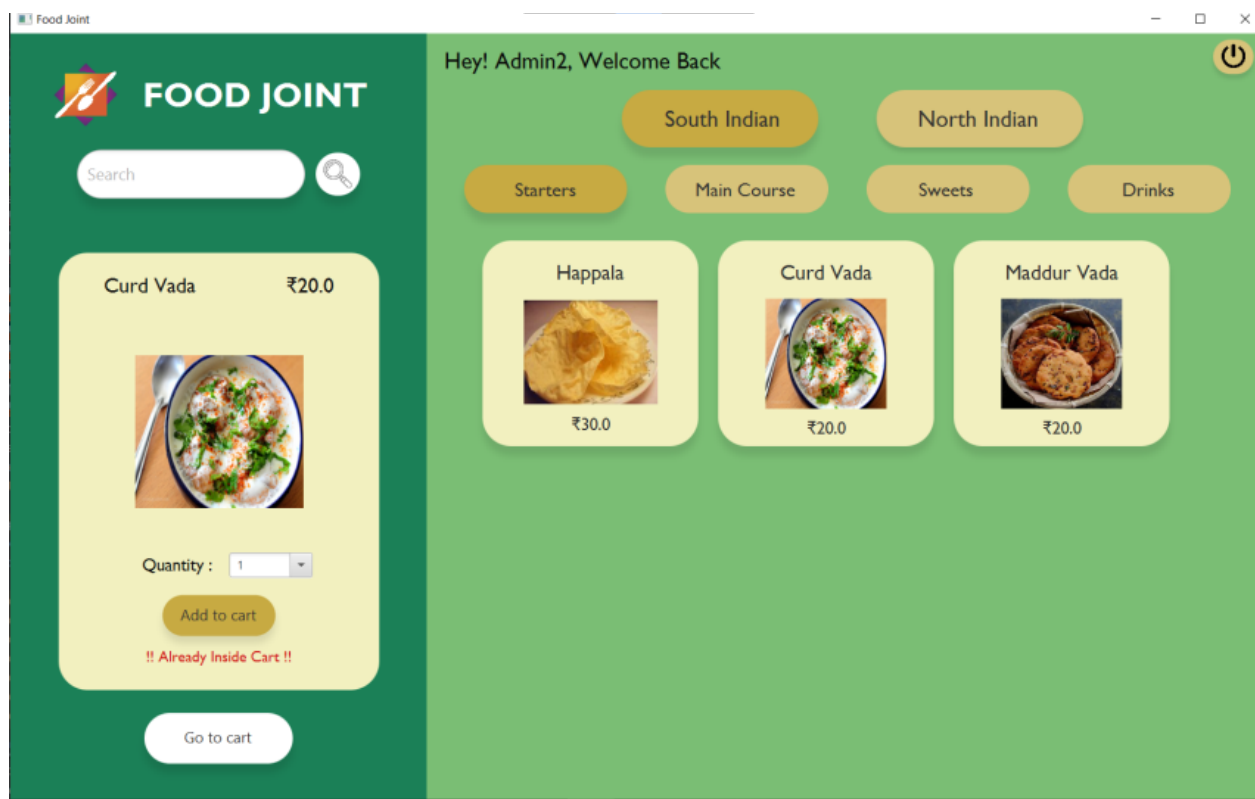
PASSWORD

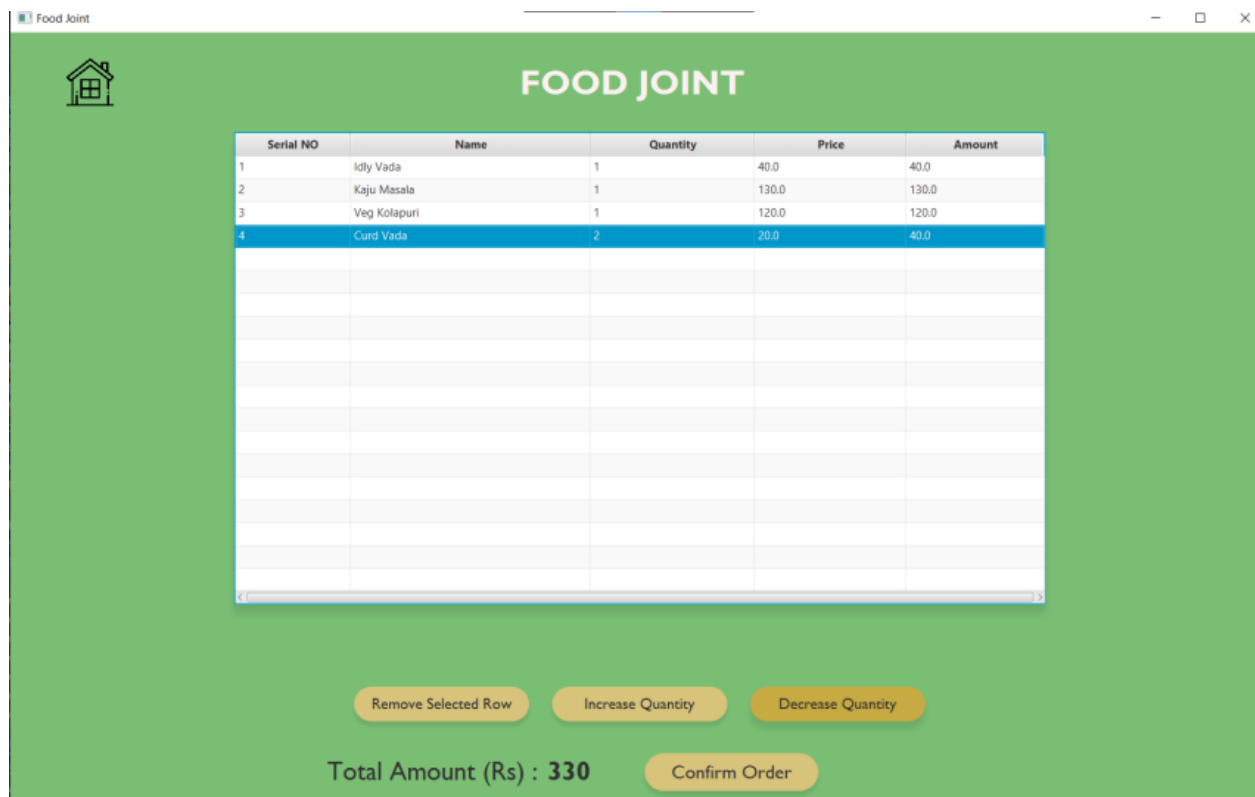
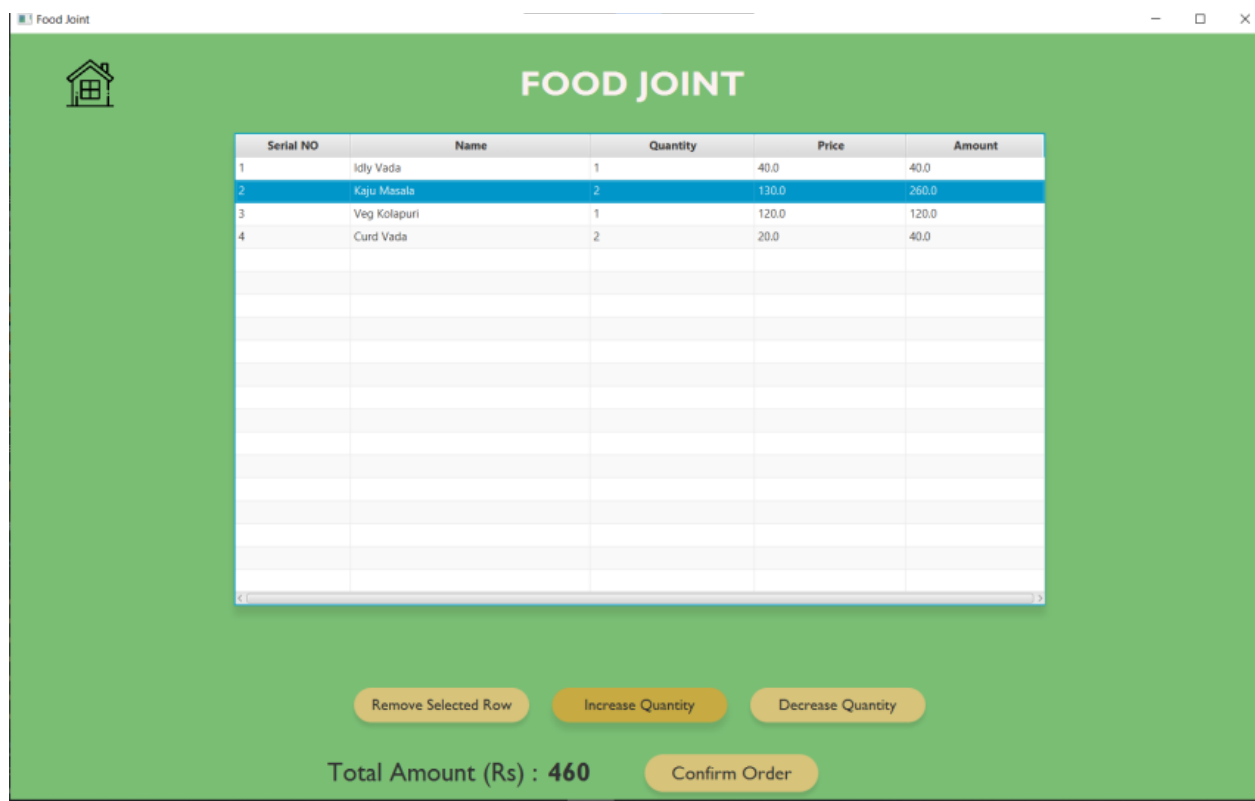
SIGN UP

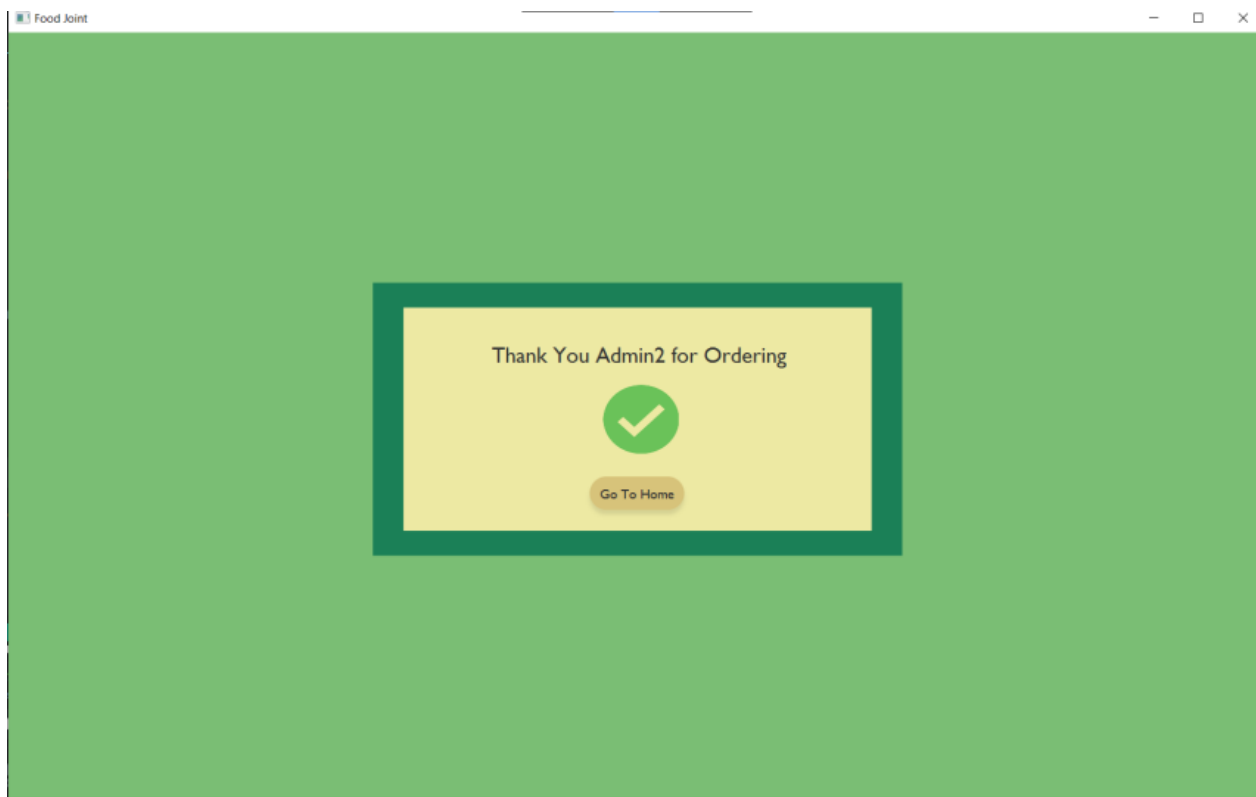
Go Back To Login Page











5. CONCLUSION

We have successfully implemented most of the java concepts. The food ordering system that we have implemented uses a local database for login and storing users info. Regex, lambda expression has been used for the user Registration process. To make it user friendly we have implemented many error detection methods and a suitable message is displayed when any such error occurs. The object oriented approach has also helped during the development. With the help of JavaFX we were able to develop appealing and interactive applications. Many javafx elements like buttons, TextField, labels, scrollpane, TableView, Gridpane, imageView and many more elements have become handy. We used SceneBuilder to design the layout and elements of the GUI. The application which we have developed is only client-sided, i.e., users can order food. So there is scope to extend this application where restaurant owners or sellers can manage inventory. We surely accept that many improvements can be made. In our application we are fetching product details from a csv file. Database service can be extended to product details also.

6. REFERENCES

- 1) [Traditional vs Object Oriented Software Development Approach](#)
- 2) [OOP - Grady Booch](#)
- 3) [JAVA - The Complete Reference - Herbert Schildt](#)
- 4) [JavaFX framework description: main features, history, requirements](#)
- 5) [JavaFX Software and System Requirements](#)
- 6) [Java - String Class and its methods explained with examples](#)
- 7) [Object-Oriented Analysis, Design, and Programming \(OOA/OOD/OOP\)](#)