NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

A MINI PROJECT REPORT

*for*

**Mini Project (21CSE46A)**

*on*

# Pizza Ordering System

*Submitted by*

**Chris Jordan**
**Adrian Mathew Aloysius**
**USN: 1NH21CS062,1NH21CS011; Sem-Sec: 4-A**

*In partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

**Academic Year: 2022-23 (EVEN SEM)**

NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

# CERTIFICATE

This is to certify that the mini project work titled

**Pizza Ordering System**

submitted in partial fulfillment of the degree of Bachelor of Engineering in Computer Science and Engineering by

**Chris Jordan**
**Adrian Mathew Aloysius**
**USN:1NH21CS062,**
**1NH21CS011**

*DURING*

*EVEN SEMESTER 2022-2023*

*for*

*Course: Mini Project -21CSE46A*

Signature of Course Instructor

## ABSTRACT

The "Pizza Application" is a Java Swing-based graphical user interface designed to facilitate the process of ordering customized pizzas. Utilizing a well-organized layout, the interface presents various options like radio buttons and checkboxes to order pizzas.

The application employs event handling and validation techniques to ensure accurate processing of user inputs, offering real-time feedback and error notifications. Exception handling mechanisms promptly address invalid inputs and guide users to correct their selections. Overall, the application demonstrates effective user interface design and programming practices, delivering a seamless and user-friendly experience for customizing and placing pizza orders.

# TABLE OF CONTENTS

# 1.0 INTRODUCTION

## 1.1 PROBLEM DEFINITION

A Pizza Shop needs a simple interface that takes the customer's pizza order based on size of the pizza, the toppings to be added and quantity of pizza. After ordering the customer's bill is displayed along with tax. The same customer has the option to place multiple orders and the bill gets updated accordingly. Once a customer has finished ordering a new customer can be added and the screen gets refreshed.

## 1.2 OBJECTIVES

Here is a list of tasks that need to be accomplished –

1. User-Friendly Interface: Create an intuitive Graphical user interface that allows users to interact with the application easily like enabling users to select pizza size using radio buttons and toppings using checkboxes, providing a seamless interaction flow.

2. Pizza Customization: Enable users to customize their pizza orders by selecting pizza sizes (small, medium, large) and a variety of toppings.

3. Real-Time Calculation: Implement a system to calculate the total cost of the order based on the selected pizza size and toppings, and display it to the user.

4. Order Validation: Validate user inputs to ensure that the number of pizzas ordered is a positive integer greater than zero.

5. Dynamic Updates: Update the interface in real-time to display the selected pizza size, toppings, toppings count, and total price.

6. Sales Tax Calculation: Calculate and display the applicable sales tax for the order.

7. Order Summary: Provide users with a detailed summary of their order, including selected pizza size, toppings, toppings count, number of pizzas, total price, and grand total.

8. Multiple Actions: Implement functionality for various user actions, such as calculating the price, placing another order, starting a new customer order, and exiting the application.

9. Error Handling: Handle exceptions and display appropriate error messages when users provide incorrect or incomplete information.

10. Code Structure: Organize the code into distinct classes and methods, promoting modularity and maintainability.

11. Testing and Debugging: Thoroughly test the application to identify and resolve any bugs, errors, or unexpected behavior.

12. Code Reusability: Design the codebase with a modular approach to facilitate potential future enhancements or feature additions.

## 1.3 METHODOLOGY

The project uses Java’s Swing and AWT library to create graphical user interface components. The components used are labels, radio buttons, checkboxes, text fields, buttons and text areas. The design includes arranging the UI components in panels, selecting appropriate colors and using layout managers to set up the structure of the main window. Event Handling mechanisms like ActionEvent and ActionListener are used to define the behavior for different button actions and handle user inputs and interactions.

## 1.4 REQUIREMENT SPECIFICATIONS

### Hardware Requirements:

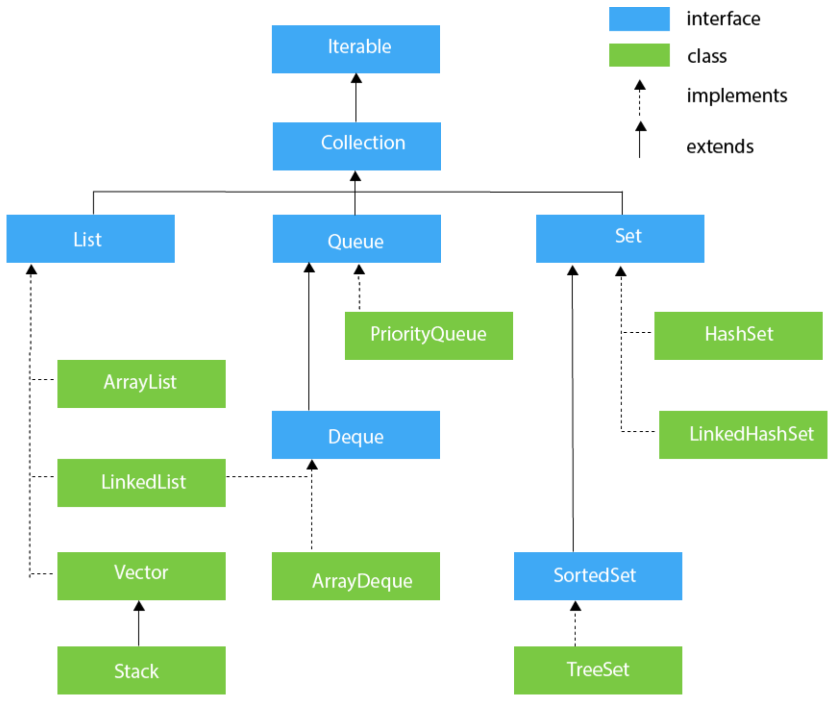
Processor: Any modern processor should be sufficient.
Memory (RAM): At least 2 GB of RAM is recommended.
Storage: The application itself is very small, so minimal storage space is required.
Display: A monitor with a resolution of 1024x768 or higher is recommended for a comfortable viewing experience.

### Software Requirements:

Java Development Kit (JDK): You'll need a compatible JDK installed on your system to compile and run Java programs. Preferably JDK 17 as it’s the LTE version.
Integrated Development Environment (IDE): While you can use a simple text editor and command-line tools to compile and run the Java program, using an IDE can greatly simplify development and debugging. Popular options include Eclipse, IntelliJ IDEA, and NetBeans.
Java Swing Library: The Java Swing library is included with the standard JDK distribution, so you don't need to install anything separately.
Operating System: The provided code should work on various operating systems, including Windows, macOS, and Linux.

## 2.0 JAVA COLLECTIONS FRAMEWORK

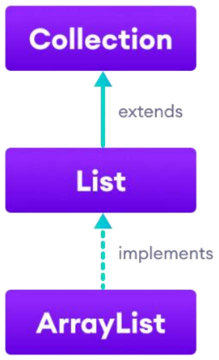
The collections in java provide an architecture to store and manipulate the group of objects, interfaces and classes. A collection is a group of objects or it is a single entity that represents multiple objects. Java collection framework consists of interfaces and classes such as Queue, Set, List, Deque and PriorityQueue, HashSet, ArrayList, Vector, LinkedList, LinkedHashSet. By using these classes and interface developers can represent a group of objects in a single entity. This framework has several useful functions that have tons of useful functions, making a programmer task super easy.



## 2.1 ArrayList

Java ArrayList is a part of the Java collection framework and it is a class of java.util package. It provides a way to store and manipulate a collection of elements in a flexible and resizable container. Unlike traditional arrays, ArrayList automatically handles resizing itself when elements are added or removed, making it convenient to use. ArrayList supports generics, which means you can specify the type of elements it will hold. For example, you can create an ArrayList to hold strings, integers, custom objects, or any other data type. ArrayList maintains the order of elements based on the order they were added. The order of elements is preserved when iterating through the list. Elements in an ArrayList can be accessed using an index, similar to arrays. This allows for efficient random access to elements.

It implements the List interface of the collections framework.



### Creating an ArrayList

Before using ArrayList, we need to import the java.util.ArrayList package first. Here is how we can create Arraylists in Java:

```
ArrayList<Type> arrayList= new ArrayList<>();
```

Here, Type indicates the type of an arraylist. For example,

```
// create Integer type arraylist
ArrayList<Integer> arrayList = new ArrayList<>();

// create String type arraylist
ArrayList<String> arrayList = new ArrayList<>();
```

In the above program, we have used Integer not int. It is because we cannot use primitive types while creating an arraylist. Instead, we have to use the corresponding wrapper classes.

Here, Integer is the corresponding wrapper class of int.

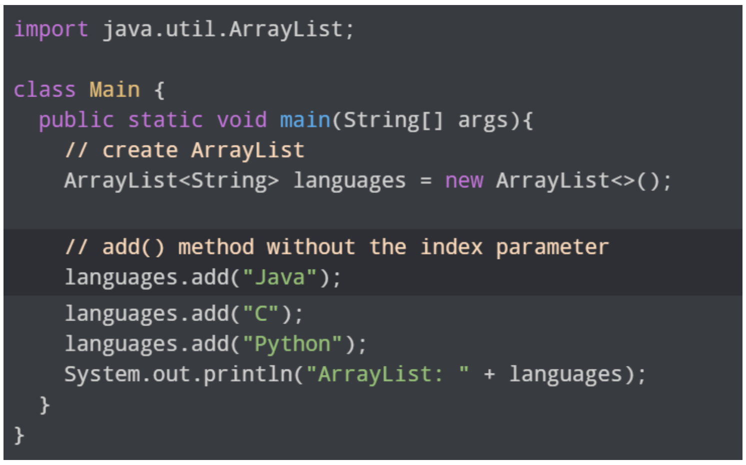
## Basic Operations on ArrayList

The ArrayList class provides various methods to perform different operations on arraylists

### 1. Add Elements to an ArrayList

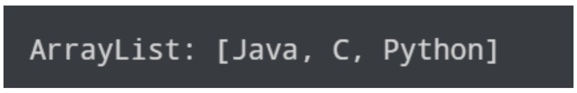
To add a single element to the arraylist, we use the add() method of the ArrayList class.

In the given example, we have created an ArrayList named languages. Here, we have used the add() method to add elements to languages.



```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args){
    // create ArrayList
    ArrayList<String> languages = new ArrayList<>();

    // add() method without the index parameter
    languages.add("Java");
    languages.add("C");
    languages.add("Python");
    System.out.println("ArrayList: " + languages);
  }
}
```

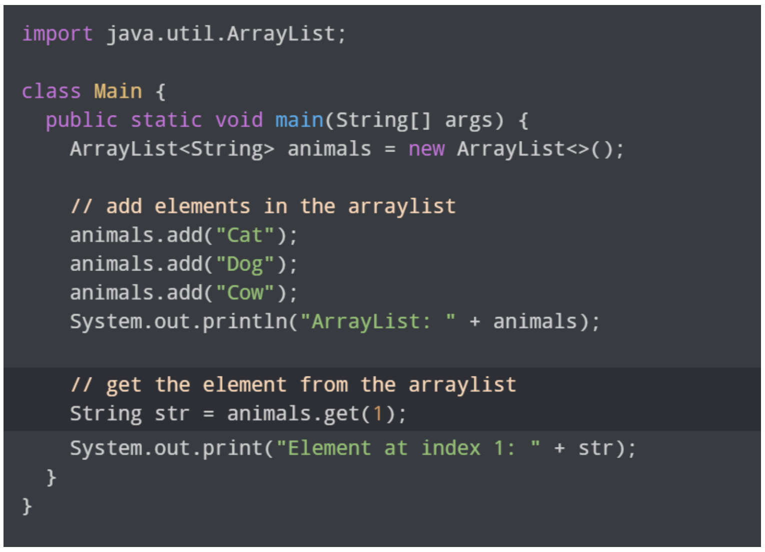
## Output



```
ArrayList: [Java, C, Python]
```

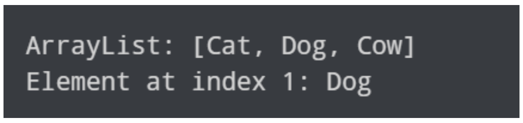
## 2. Access ArrayList Elements

To access an element from the arraylist, we use the get() method of the ArrayList class. In the given example, we have used the get() method with parameter 1. Here, the method returns the element at index 1.



```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args) {
    ArrayList<String> animals = new ArrayList<>();

    // add elements in the arraylist
    animals.add("Cat");
    animals.add("Dog");
    animals.add("Cow");
    System.out.println("ArrayList: " + animals);

    // get the element from the arraylist
    String str = animals.get(1);
    System.out.print("Element at index 1: " + str);
  }
}
```

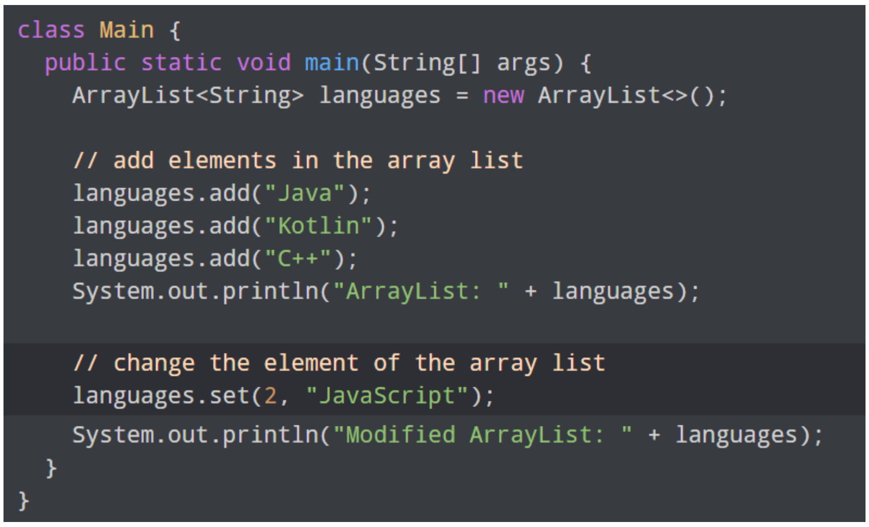
## Output



```
ArrayList: [Cat, Dog, Cow]
Element at index 1: Dog
```

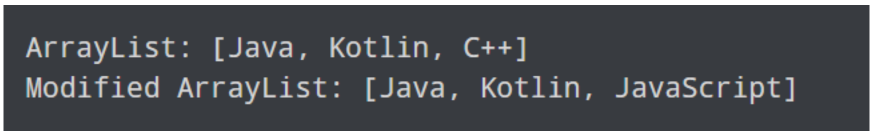
## 3. Change ArrayList Elements

To change elements of the arraylist, we use the set() method of the ArrayList class. In the given example, we have created an ArrayList named languages, Here, the set() method changes the element at index 2 to JavaScript.



```
class Main {
  public static void main(String[] args) {
    ArrayList<String> languages = new ArrayList<>();

    // add elements in the array list
    languages.add("Java");
    languages.add("Kotlin");
    languages.add("C++");
    System.out.println("ArrayList: " + languages);

    // change the element of the array list
    languages.set(2, "JavaScript");
    System.out.println("Modified ArrayList: " + languages);
  }
}
```

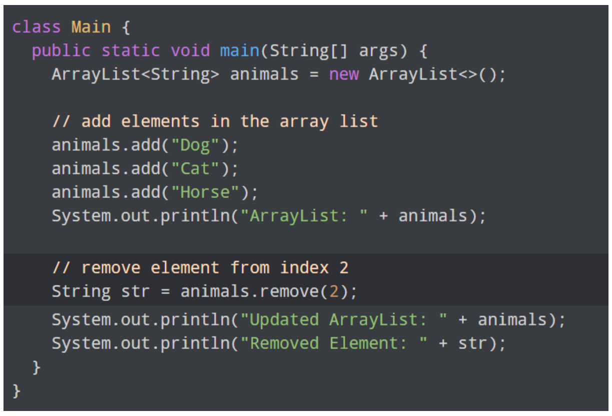
**Output**



```
ArrayList: [Java, Kotlin, C++]
Modified ArrayList: [Java, Kotlin, JavaScript]
```

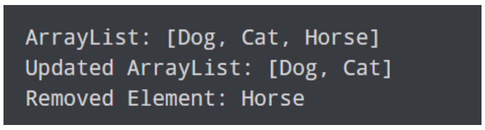
## 4. Remove ArrayList Elements

To remove an element from the arraylist, we can use the remove() method of the ArrayList class. Here, the remove() method takes the index number as the parameter. And, removes the element specified by the index number.



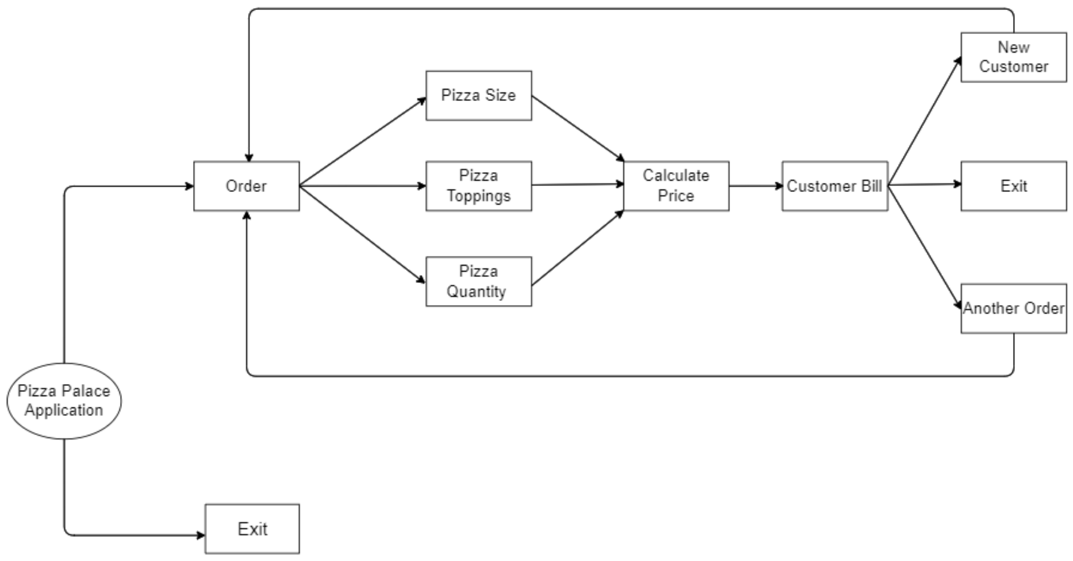
```
class Main {
  public static void main(String[] args) {
    ArrayList<String> animals = new ArrayList<>();

    // add elements in the array list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
    System.out.println("ArrayList: " + animals);

    // remove element from index 2
    String str = animals.remove(2);
    System.out.println("Updated ArrayList: " + animals);
    System.out.println("Removed Element: " + str);
  }
}
```

**Output**

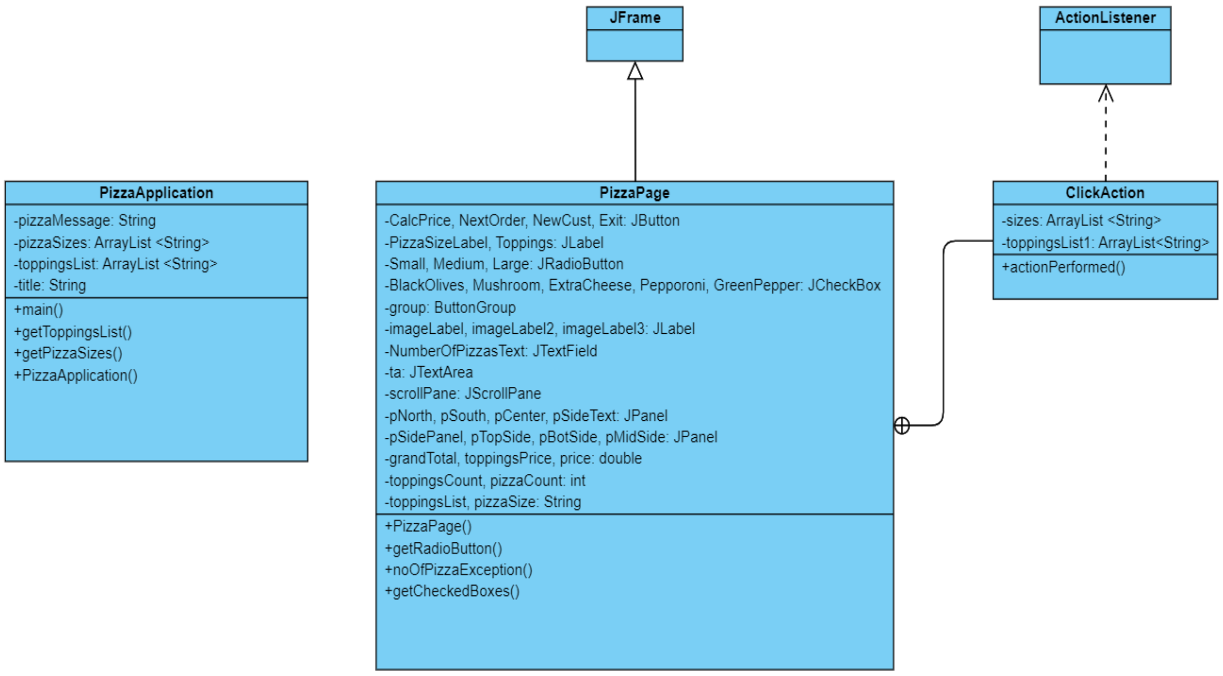


```
ArrayList: [Dog, Cat, Horse]
Updated ArrayList: [Dog, Cat]
Removed Element: Horse
```

## 3.1 FUNCTIONAL BLOCK DIAGRAM



Block Diagram


JFrame
ActionListener
PizzaApplication
-pizzaMessage: String
-pizzaSizes: ArrayList <String>
-toppingsList: ArrayList <String>
-title: String
+main()
+getToppingsList()
+getPizzaSizes()
+PizzaApplication()
PizzaPage
-CalcPrice, NextOrder, NewCust, Exit: JButton
-PizzaSizeLabel, Toppings: JLabel
-Small, Medium, Large: JRadioButton
-BlackOlives, Mushroom, ExtraCheese, Pepporoni, GreenPepper: JCheckBox
-group: ButtonGroup
-imageLabel, imageLabel2, imageLabel3: JLabel
-NumberOfPizzasText: JTextField
-ta: JTextArea
-scrollPane: JScrollPane
-pNorth, pSouth, pCenter, pSideText: JPanel
-pSidePanel, pTopSide, pBotSide, pMidSide: JPanel
-grandTotal, toppingsPrice, price: double
-toppingsCount, pizzaCount: int
-toppingsList, pizzaSize: String
+PizzaPage()
+getRadioButton()
+noOfPizzaException()
+getCheckedBoxes()
ClickAction
-sizes: ArrayList <String>
-toppingsList1: ArrayList<String>
+actionPerformed()

Class Diagram

## 3.2 MODULE DESCRIPTION

### 1) PizzaApplication:

This class serves as the entry point for the application. It defines the main method where the GUI window (PizzaPage) is instantiated and displayed.

It initializes the application title and defines strings for messages related to pizza sizes.

### Data Initialization:

This class contains two ArrayLists: pizzaSizes and toppingsList, which store predefined pizza sizes and toppings.

It initializes predefined pizza sizes and toppings using ArrayLists within its constructor.

It encapsulates these lists and provides getter methods to access them from other parts of the application.

### 2) PizzaPage:

This class represents the main graphical interface of the application, extending the JFrame class.

It consists of various Swing components such as buttons, labels, radio buttons, checkboxes, and text fields to interact with the user.

The GUI is divided into three main sections: North, Center and South.

The North section displays images and the application title.

The Center section contains panels for selecting pizza size and toppings.

The South section contains buttons for actions like calculating the price, placing another order, starting a new customer order, and exiting.

It includes an ActionListener (ClickAction) that handles user interactions and implements the logic for calculating prices, handling exceptions, and managing the user's selections.

### 3) ClickAction:

This inner class of PizzaPage implements the ActionListener interface to respond to button clicks and other events. It handles actions such as calculating the pizza price, handling exceptions for input validation, and resetting fields for a new order.

It communicates with the PizzaApplication class to retrieve predefined sizes and toppings.

## Functionality:

Users can select a pizza size (Small, Medium, or Large) using radio buttons.

Users can select toppings (Black Olives, Mushroom, Extra Cheese, Pepperoni, Green Pepper) using checkboxes.

Users can enter the number of pizzas they want to order.

Clicking the "Calculate Price" button calculates the total price based on the selected options and displays it, along with toppings and taxes.

Clicking the "Another Order" button clears the selections for a new order.

Clicking the "New Customer" button clears the selections and the displayed order history.

Clicking the "Exit" button closes the application.

**Advantages:**

The application provides an interactive and user-friendly way to order pizzas with various options.

The use of ArrayLists and the Java Collections Framework makes it easy to manage and maintain predefined sizes and toppings.

The application's modular structure promotes code organization and reusability.

# 4.0 IMPLEMENTATION

## 4.1 PizzaApplication Class (Main Class)

```
public class PizzaApplication {

   public  static String title = "Welcome to Pizza Palace";

   public  static String pizzaSizeMessage="Please Choose a Pizza Size!!!";

   public  static String pizzaSizeMessageWindow="You Didn't Choose a Size!!!";

   public  static String noOfPizzasMessage="Please enter the number of Pizzas!!!";

   public  static String toppings="Toppings:";

   ArrayList<String> pizzaSizes = new ArrayList<String>();

   ArrayList<String> toppingsList = new ArrayList<String>();

   public PizzaApplication() {

      pizzaSizes.add("Number of Pizzas");

      pizzaSizes.add("Pizza Size");

      pizzaSizes.add("Regular");

      pizzaSizes.add("Small");

      pizzaSizes.add("Medium");

      pizzaSizes.add("Large");

      toppingsList.add("Black Olives");

      toppingsList.add("Mushroom");

      toppingsList.add("Extra Cheese");

      toppingsList.add("Pepperoni");

      toppingsList.add("GreenPepper");

   }

   public ArrayList<String> getToppingsList() {

      return toppingsList;

   }
```

```
    public ArrayList<String> getPizzaSizes() {
        return pizzaSizes;
    }
    public static void main (String[] args) {
        PizzaPage frame=new PizzaPage();
        frame.setTitle(title);
        frame.setSize(600,600);
        frame.setResizable(true);
        frame.setVisible(true);
        /*Center the window*/
        frame.setLocationRelativeTo(null);
    }
}
```

## 4.2 PizzaPage Class

### 4.2.1 Components Declaration and GUI Layout

```
class PizzaPage extends JFrame {
    private static final long serialVersionUID = 1L;
    /*Declare Components to be used in the PizzaPage*/
    private JButton CalcPrice, NextOrder,NewCust,Exit;
    private JLabel PizzaSizeLabel,Toppings;
    private JRadioButton Small, Medium, Large;
    private JCheckBox BlackOlives,Mushroom,ExtraCheese,Pepporoni,GreenPepper;
    private ButtonGroup group;
    private JLabel imageLabel,imageLabel2,imageLabel3;
    private JTextField NumberOfPizzasText;
    private JTextArea ta;
    private JScrollPane scrollPane;

    private JPanel pNorth, pSouth, pCenter, pSideText;

    private JPanel pSidePanel,pTopSide,pBotSide,pMidSide;

    /*Temporary Variables*/

    private double grandTotal=0;

    ImageIcon image;

    private double price;

    private int toppingsCount,pizzaCount;

    private double toppingsPrice;

    private String toppingsList;

    private String pizzaSize;

    public PizzaPage()

    {

      PizzaApplication papp = new PizzaApplication();

      ArrayList<String> sizes = papp.getPizzaSizes();

      ArrayList<String> tops = papp.getToppingsList();

      toppingsList=sizes.get(2);

      grandTotal=0;

      setDefaultCloseOperation(EXIT_ON_CLOSE);

      Container window=getContentPane();

      window.setLayout(new BorderLayout(10,10));

      NumberOfPizzasText=new JTextField("0",2);

      NumberOfPizzasText.setEditable(true);

      NumberOfPizzasText.setToolTipText(sizes.get(0));

      ta=new JTextArea("",20,20);

      ta.setEditable(false);
```

```
scrollPane=new JScrollPane(ta);

pNorth=new JPanel();

pNorth.setLayout(new GridLayout(1,3));

pCenter=new JPanel();

pCenter.setLayout(new GridLayout(1,2));

pTopSide=new JPanel();

pTopSide.setLayout(new GridLayout(1,4));

pTopSide.setBorder(BorderFactory.createMatteBorder(

        0, 0, 5, 0, Color.gray));

pMidSide=new JPanel();

pMidSide.setLayout(new GridLayout(6,1));

pBotSide=new JPanel();

pBotSide.setLayout(new GridLayout(1,4));

pBotSide.setBorder(BorderFactory.createMatteBorder(

        5, 0, 0, 0, Color.gray));

pSouth=new JPanel();

pSouth.setLayout(new GridLayout(1,4));

Small =new JRadioButton(sizes.get(3));

Medium =new JRadioButton(sizes.get(4));

Large =new JRadioButton(sizes.get(5));

group=new ButtonGroup();

group.add(Small);

group.add(Medium);

group.add(Large);

imageLabel=new JLabel();

imageLabel2=new JLabel();
```

```
imageLabel3=new JLabel();

imageLabel.setIcon(image);

imageLabel.setOpaque(true);

imageLabel.setBackground(Color.WHITE);

imageLabel2.setText("<html><FONT SIZE=10>"+PizzaApplication.title+"</FONT></html>");

imageLabel2.setOpaque(true);

imageLabel2.setBackground(Color.WHITE);

imageLabel3.setIcon(image);

imageLabel3.setOpaque(true);

imageLabel3.setBackground(Color.WHITE);

pNorth.add(imageLabel);

pNorth.add(imageLabel2);

pNorth.add(imageLabel3);

pSideText=new JPanel();

pSideText.setLayout(new GridLayout(1,1));

pSideText.add(scrollPane);

Toppings=new JLabel(PizzaApplication.toppings);

BlackOlives=new JCheckBox(tops.get(0));

Mushroom=new JCheckBox(tops.get(1));

ExtraCheese=new JCheckBox(tops.get(2));

Pepporoni=new JCheckBox(tops.get(3));

GreenPepper=new JCheckBox(tops.get(4));

pMidSide.add(Toppings);

pMidSide.add(BlackOlives);

pMidSide.add(Mushroom);

pMidSide.add(ExtraCheese);
pMidSide.add(Pepporoni);

pMidSide.add(GreenPepper);

pBotSide.add(new JLabel(sizes.get(0)));

pBotSide.add(NumberOfPizzasText);

PizzaSizeLabel=new JLabel(sizes.get(1));

pTopSide.add(PizzaSizeLabel);

pTopSide.add(Small);

pTopSide.add(Medium);

pTopSide.add(Large);

pSidePanel=new JPanel();

pSidePanel.setLayout(new BorderLayout(10,10));

pSidePanel.add("North",pTopSide);

pSidePanel.add("Center",pMidSide);

pSidePanel.add("South",pBotSide);

CalcPrice = new JButton("Calculate Price");

NextOrder =new JButton("Another Order");

NewCust=new JButton("New Customer");

Exit=new JButton("Exit");

pSouth.add(CalcPrice);

pSouth.add(NextOrder);

pSouth.add(NewCust);

pSouth.add(Exit);

ClickAction handler=new ClickAction();

CalcPrice.addActionListener(handler);

NewCust.addActionListener(handler);

NextOrder.addActionListener(handler);
```

```
    Exit.addActionListener(handler);
    pCenter.add(pSidePanel);
    pCenter.add(pSideText);
    window.add("North",pNorth);
    window.add("Center",pCenter);
    window.add("South",pSouth);
  }
```

### 4.2.2 ClickAction Class (Event Handling)

```
  private class ClickAction implements ActionListener
  {
    PizzaApplication papp = new PizzaApplication();
    ArrayList<String> sizes = papp.getPizzaSizes();
    ArrayList<String> toppingsList1 = papp.getToppingsList();
    boolean clearFields=false;
    public void actionPerformed(ActionEvent event)
    {
      if(event.getSource()==CalcPrice)
      {
        getRadioButton();
        getCheckedBoxes();
        try
        {
          pizzaCount=Integer.valueOf(NumberOfPizzasText.getText());
          if(pizzaCount<1){
            noOfPizzaException();
            clearFields=true;
```

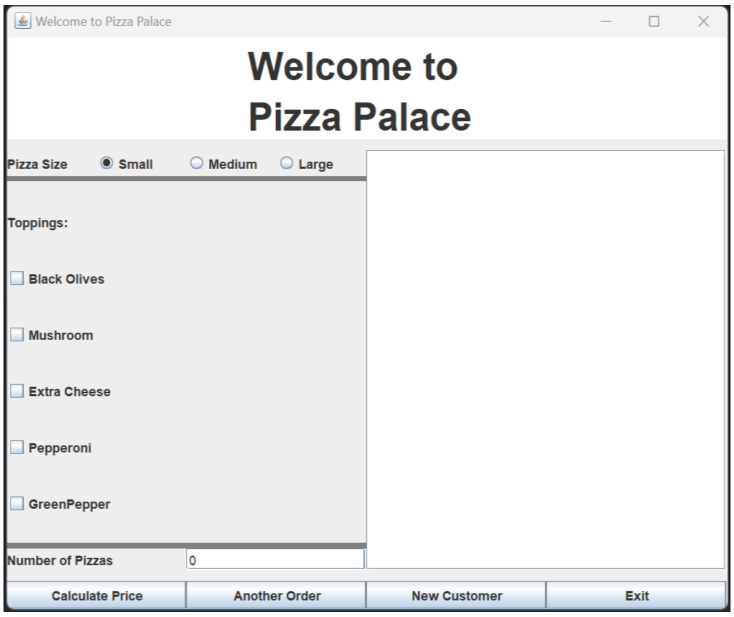
```
}
else if(pizzaCount>0)
{
   ta.append(sizes.get(1)+": ");
   ta.append(pizzaSize+"($"+price+")"+"\n");
   ta.append("Price of each Pizza without Tax: ");
   if(toppingsCount>=1)
   {
      toppingsPrice = toppingsCount * 0.6;
      price+=toppingsPrice;
   }
   ta.append(price+"\n");
   ta.append("Number of Toppings: ");
   ta.append(toppingsCount+"\n");
   ta.append("Toppings: ");
   ta.append(toppingsList+"\n");
   ta.append("NumberOfPizzas: ");
   ta.append(pizzaCount+"\n");
   ta.append("SalesTax: ");
   ta.append(((0.08*price*pizzaCount))+"\n");
   ta.append("Total: ");
   price=((0.08*price*pizzaCount)+price*pizzaCount);
   ta.append(price+"\n\n");
   grandTotal+=price;
   ta.append("Grand Total: $");
   ta.append(grandTotal+"\n\n");
```

```
            }
        }
        catch(Exception e)
        {
            JPanel warning=new JPanel();
            JOptionPane.showMessageDialog
                    (warning,
                            "Please Enter a Positive Integer more than 0 in the Number of Pizzas Field!!!",
                            "Not a Integer Problem!!!",
                            JOptionPane.ERROR_MESSAGE
                    );
            clearFields= true;
        }
        if(!clearFields) {
            group.clearSelection();
            BlackOlives.setSelected(false);
            Mushroom.setSelected(false);
            ExtraCheese.setSelected(false);
            Pepporoni.setSelected(false);
            GreenPepper.setSelected(false);
            NumberOfPizzasText.setText("");
        }
    }
    if(event.getSource()==Exit)
    {
        System.exit(0);
        }
        if(event.getSource()== NextOrder)
        {
            group.clearSelection();
            BlackOlives.setSelected(false);
            Mushroom.setSelected(false);
            ExtraCheese.setSelected(false);
            Pepporoni.setSelected(false);
            GreenPepper.setSelected(false);
            NumberOfPizzasText.setText("");
        }
        if(event.getSource()==NewCust)
        {
            ta.setText("");
            grandTotal=0;
            group.clearSelection();
            BlackOlives.setSelected(false);
            Mushroom.setSelected(false);
            ExtraCheese.setSelected(false);
            Pepporoni.setSelected(false);
            GreenPepper.setSelected(false);
            NumberOfPizzasText.setText("");
        }
    }
}
```

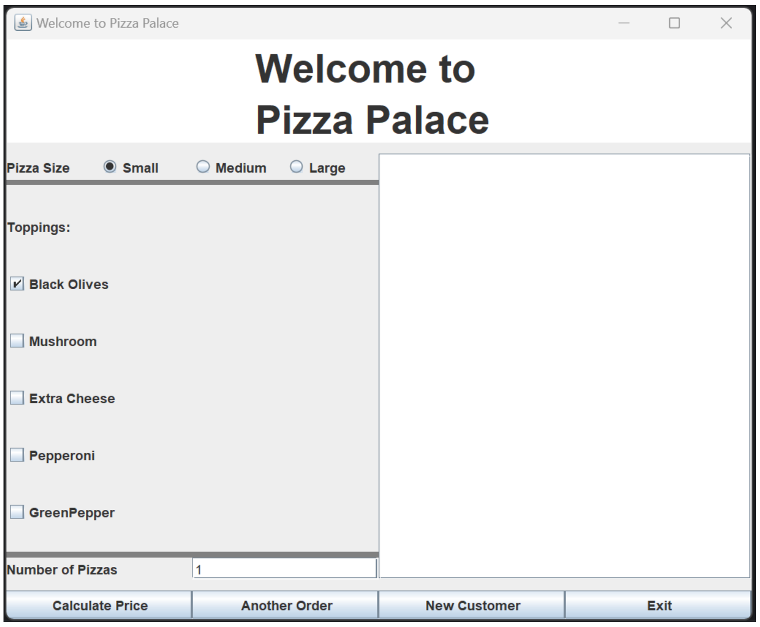
### 4.2.3 Helper Functions

```
private void getRadioButton()
  {
     price = 0.0;
     if(Small.isSelected())
     {
        pizzaSize="Small";
        price = 12.0;
     }
     else if(Medium.isSelected())
     {
        pizzaSize="Medium";
        price = 14.0;
     }
     else if(Large.isSelected())
     {
        pizzaSize="Large";
        price = 16.0;
     }
  }
  private void noOfPizzaException(){
     PizzaApplication papp = new PizzaApplication();
     ArrayList<String> sizes = papp.getPizzaSizes();
     JPanel warning=new JPanel();
     JOptionPane.showMessageDialog
          (warning,
                PizzaApplication.noOfPizzasMessage,
                sizes.get(0),
                JOptionPane.ERROR_MESSAGE
            );
    }
    private void getCheckedBoxes()
    {
        PizzaApplication papp = new PizzaApplication();
        ArrayList<String> toppingsList1 = papp.getToppingsList();
        toppingsCount=0;
        toppingsList="Regular";
        if(price==0.0)
        {
            JPanel warning=new JPanel();
            JOptionPane.showMessageDialog
                (warning,
                    PizzaApplication.pizzaSizeMessage,
                    PizzaApplication.pizzaSizeMessageWindow,
                    JOptionPane.ERROR_MESSAGE
                );
        }
        else
        {
            if(BlackOlives.isSelected())
            {
                toppingsCount++;
                toppingsList=toppingsList+", "+toppingsList1.get(0);
            }
            if(Mushroom.isSelected())
            {
                toppingsCount++;
                toppingsList=toppingsList+", "+toppingsList1.get(1);
            }
            if(ExtraCheese.isSelected())
            {
                toppingsCount++;
                toppingsList=toppingsList+", "+toppingsList1.get(2);
            }
            if(Pepporoni.isSelected())
            {
                toppingsCount++;
                toppingsList=toppingsList+", "+toppingsList1.get(3);
            }
            if(GreenPepper.isSelected())
            {
                toppingsCount++;
                toppingsList=toppingsList+", "+toppingsList1.get(4);
            }
        }
    }
```

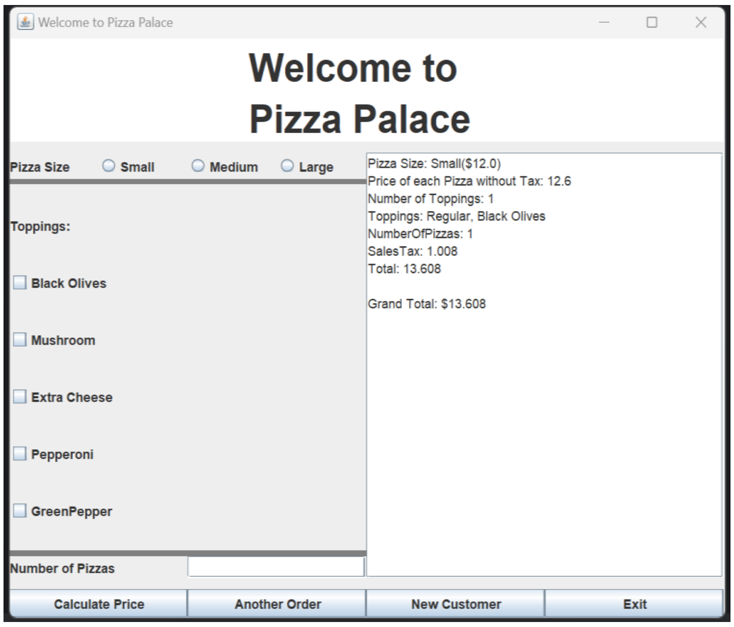
## 5.0 RESULTS



**Output-1 - Start**

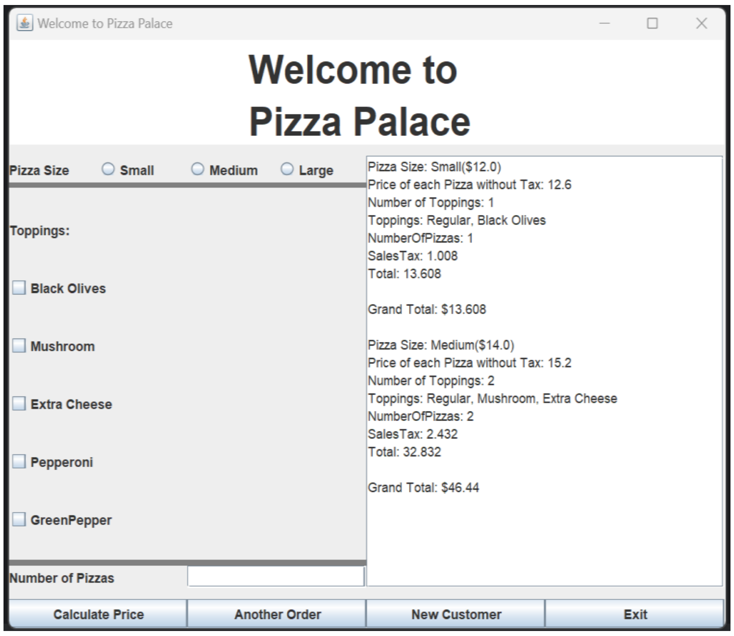


**Output-2 - Order**



```
Pizza Size: Small($12.0)
Price of each Pizza without Tax: 12.6
Number of Toppings: 1
Toppings: Regular, Black Olives
NumberOfPizzas: 1
SalesTax: 1.008
Total: 13.608

Grand Total: $13.608
```

**Output 3 - Bill**



```
Pizza Size: Small($12.0)
Price of each Pizza without Tax: 12.6
Number of Toppings: 1
Toppings: Regular, Black Olives
NumberOfPizzas: 1
SalesTax: 1.008
Total: 13.608

Grand Total: $13.608

Pizza Size: Medium($14.0)
Price of each Pizza without Tax: 15.2
Number of Toppings: 2
Toppings: Regular, Mushroom, Extra Cheese
NumberOfPizzas: 2
SalesTax: 2.432
Total: 32.832

Grand Total: $46.44
```

**Output 4 – Another Order**

## 6.0 CONCLUSION

In conclusion, the "Pizza Application" exemplifies a well-designed Java Swing-based graphical user interface that simplifies the process of ordering customized pizzas. Through an intuitive layout, the user interface is divided into distinct sections, each serving a specific purpose—providing an engaging visual experience. The application utilizes ArrayLists to manage predefined pizza sizes and toppings, contributing to its extensibility and maintainability.

The "Pizza Application" showcases effective event handling and validation techniques, ensuring that user inputs are accurately processed and errors are gracefully handled. The implementation of radio buttons for pizza sizes and checkboxes for toppings offers a streamlined and intuitive selection process. Additionally, the application calculates and presents the total cost of the order, accounting for the chosen toppings, pizza size, quantity, and sales tax.

Exception handling is well-incorporated throughout the code, promptly notifying users of invalid inputs and guiding them to correct their mistakes. The modular design, encapsulated within classes, promotes code organization and readability, making it easier to understand and maintain the application's functionality.

In summary, the "Pizza Application" stands as a testament to effective user interface design and Java programming principles. Its user-friendly design, offers a seamless and enjoyable experience for users looking to place pizza orders with customization options.

## 7.0 REFERENCES

1. Swing Components - https://www.tutorialspoint.com/swing/swing_components.htm

   https://www.geeksforgeeks.org/jradiobutton-in-java/

2. Event Handling - https://docs.oracle.com/javase/tutorial/uiswing/events/

3. Java Collections –

   https://www.geeksforgeeks.org/arraylist-in-java/

   https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html