```
/*******************************************************************************************
 *  This class hold information regarding a pizza order
 *
 *  The user is to enter the following data for a pizza order:
 *      pizza specialty
 *      server name
 *      delivery method
 *      pizza size
 *      optional extra ingredients
 *      spiciness level
 *      optional special instructions
 *
 *  CST 183 Programming Assignment 8
 *  @author Michael Clinesmith
 *******************************************************************************************/
public class PizzaOrder
{

    private ItemElement pizzaSpecialty, serverName, orderType, pizzaSize,
                        pizzaECOption, pizzaESOption, pizzaGCOption, pizzaSCOption,
                        spiciness, specialInstructions;

    /**
     * Basic constructor that sets up blank options
     */
    public PizzaOrder()
    {
        pizzaSpecialty = new ItemElement();
        serverName = new ItemElement();
        orderType = new ItemElement();
        pizzaSize = new ItemElement();

        pizzaECOption = new ItemElement();
        pizzaESOption = new ItemElement();
        pizzaGCOption = new ItemElement();
        pizzaSCOption = new ItemElement();
        spiciness = new ItemElement();
        specialInstructions = new ItemElement();

    }

    /**
     * Constructor that creates on object with the given paramaters
     * @param specialty String type of specialty pizza ordered
     * @param name String name of server
     * @param type String type of delivery
     * @param size String size of pizza
     * @param xCheese boolean if extra cheese selected
     * @param xSauce boolean if extra sauce selected
     * @param garlic boolean if garlic crust selected
     * @param stuffed boolean if stuffed crust selected
     * @param spicy double spiciness level of sauce
     * @param instruct String special instructions given
     */
    public PizzaOrder(String specialty, String name, String type, String size,  boolean xCheese, boolean xSauce,
                      boolean garlic, boolean stuffed, double spicy, String instruct )
    {
        pizzaSpecialty = new ItemElement("Pizza Selection", specialty, 0);
        serverName = new ItemElement("Server Name", name, 1);
        orderType = new ItemElement("Delivery Method", type, 1);
```

30/30 points for Program 8

GRADING

CHECKED:
- Inclusion of required GUI components
- Back-end class to store pizza data
- General capture of input
- Safe behavior of GUI with immediate click
- General coding standards for documentation/structure
- Existence and behavior of "Exit/End" button

COMMENTS:
- Awesome work! Way above-and-beyond.
- Graphics and color a very nice touch. Great embellishments to specs.
- More than meets basic program requirements.
- Some additional thoughts and comments embedded, but nothing much to complain about.

```
        pizzaSize = new ItemElement("Pizza Size", size, 1);
        if (xCheese)
        {
            pizzaECOption = new ItemElement("Extra Cheese Option", "Selected", 2);
        }
        else
        {
            pizzaECOption = new ItemElement("Extra Cheese Option", "Not Selected", 2);
        }
        if (xSauce)
        {
            pizzaESOption = new ItemElement("Extra Sauce Option", "Selected", 2);
        }
        else
        {
            pizzaESOption = new ItemElement("Extra Sauce Option", "Not Selected", 2);
        }
        if (garlic)
        {
            pizzaGCOption = new ItemElement("Garlic Crust Option", "Selected", 2);
        }
        else
        {
            pizzaGCOption = new ItemElement("Garlic Crust Option", "Not Selected", 2);
        }
        if (stuffed)
        {
            pizzaSCOption = new ItemElement("Stuffed Crust Option", "Selected", 2);
        }
        else
        {
            pizzaSCOption = new ItemElement("Stuffed Crust Option", "Not Selected", 2);
        }
        spiciness = new ItemElement("Spiciness Level", Double.toString(spicy), 1);
        specialInstructions = new ItemElement("Special Instructions", instruct, 1);

    }

    /**
     *  Setters for the pizza order individual elements are below
     */
    public void setPizzaSpecialty(String specialty)
    {
        pizzaSpecialty = new ItemElement("Pizza Selection", specialty, 0);
    }

    public void setServerName(String name)
    {
        serverName = new ItemElement("Server Name", name, 1);

    }

    public void setOrderType(String type)
    {
        orderType = new ItemElement("Delivery Method", type, 1);
    }

    public void setPizzaSize(String size)
    {
        pizzaSize = new ItemElement("Pizza Size", size, 1);
```

```
}

public void setPizzaECOption(boolean xCheese)
{
    if (xCheese)
    {
        pizzaECOption = new ItemElement("Extra Cheese Option", "Selected", 2);
    }
    else
    {
        pizzaECOption = new ItemElement("Extra Cheese Option", "Not Selected", 2);
    }
}

public void setPizzaESOption(boolean xSauce)
{
    if (xSauce)
    {
        pizzaESOption = new ItemElement("Extra Sauce Option", "Selected", 2);
    }
    else
    {
        pizzaESOption = new ItemElement("Extra Sauce Option", "Not Selected", 2);
    }
}

public void setPizzaGCOption(boolean garlic)
{
    if (garlic)
    {
        pizzaGCOption = new ItemElement("Garlic Crust Option", "Selected", 2);
    }
    else
    {
        pizzaGCOption = new ItemElement("Garlic Crust Option", "Not Selected", 2);
    }
}

public void setPizzaSCOption(boolean stuffed)
{
    if (stuffed)
    {
        pizzaSCOption = new ItemElement("Stuffed Crust Option", "Selected", 2);
    }
    else
    {
        pizzaSCOption = new ItemElement("Stuffed Crust Option", "Not Selected", 2);
    }
}

public void setSpiciness(double spicy)
{
    spiciness = new ItemElement("Spiciness Level", Double.toString(spicy), 1);
}

public void setSpecialInstructions(String instruct)
{
    specialInstructions = new ItemElement("Special Instructions", instruct, 1);
}
```

You might be able to generalize all of these methods into one "processOptions() method. Maybe an array of booleans parallel to an array of option name strings.

```
/*****************************************************************
 *
 * Individual getters still need to be set up for application

public ItemElement getPizzaSpecialty()
{
    return pizzaSpecialty;
}

public ItemElement getServerName()
{
    return serverName;
}

public ItemElement getOrderType()
{
    return orderType;
}

public ItemElement getPizzaSize()
{
    return pizzaSize;
}

public ItemElement getPizzaECOption()
{
    return pizzaECOption;
}

public ItemElement getPizzaESOption()
{
    return pizzaESOption;
}

public ItemElement getPizzaGCOption()
{
    return pizzaGCOption;
}

public ItemElement getPizzaSCOption()
{
    return pizzaSCOption;
}

public ItemElement getSpiciness()
{
    return spiciness;
}

public ItemElement getSpecialInstructions()
{
    return specialInstructions;
}

 *******************************************************/

/**
 * Method to return information stored in the object
 * @return String information about order saved
 */
```

```
    @Override
    public String toString()
    {
        String message="";
        if(pizzaSpecialty.toString().length()!=0)               // if no data return blank string
        {
            message += pizzaSpecialty.toString() + "\n";
            message += serverName.toString() + "\n";
            message += orderType.toString() + "\n";
            message += pizzaSize.toString() + "\n";
            message += pizzaECOption.toString() + "\n";
            message += pizzaESOption.toString() + "\n";
            message += pizzaGCOption.toString() + "\n";
            message += pizzaSCOption.toString() + "\n";
            message += spiciness.toString() + "\n";
            message += specialInstructions.toString() + "\n";
        }
        return message;
    }


}
```

```
/*********************************************************************************************
 *  This class holds information regarding a pizza order element
 *
 *  It contains a property, the property value and a level of indentation
 *
 *  CST 183 Programming Assignment 8
 *  @author Michael Clinesmith
 *********************************************************************************************/

public class ItemElement
{

    private String itemName;
    private String itemProperty;
    private int level;      // used to keep track of indentation level

    public ItemElement()
    {
        itemName = "";
        itemProperty = "";
        level = 0;
    }

    public ItemElement(String name)
    {
        itemName = name;
        itemProperty = "";
        level = 0;
    }

    public ItemElement(String name, String property)
    {
        itemName = name;
        itemProperty = property;
        level = 0;
    }

    public ItemElement(String name, String property, int lev)
    {
        itemName = name;
        itemProperty = property;

        if (lev < 0)
        {
            lev = 0;
        }
        level = lev;
    }

    public void setItemName(String itemName)
    {
        this.itemName = itemName;
    }

    public void setItemProperty(String itemProperty)
    {
        this.itemProperty = itemProperty;
    }

    public void setLevel(int level)
```

Great idea for a class.

```
    {
        if (level<0)
        {
            level = 0;
        }
        this.level = level;
    }

    public String getItemName()
    {
        return itemName;
    }

    public String getItemProperty()
    {
        return itemProperty;
    }

    public int getLevel()
    {
        return level;
    }

    @Override
    public String toString()
    {
        String message = "";
        if (itemName.length()!=0)                          // return empty String if no data
        {
            for (int i = 0; i < level; i++)                // gives indentation for level
            {
                message += "     ";
            }
            message += itemName + ": " + itemProperty;
        }
        return message;
    }

}
```

```
/*******************************************************************************************
 *  This program simulates a pizza order graphical interface and application
 *  and tests the funcationality of the PizzaOrder and ItemElement objects
 *
 *  The user is to enter the following data for a pizza order:
 *      pizza specialty
 *      server name
 *      delivery method
 *      pizza size
 *      optional extra ingredients
 *      spiciness level
 *      optional special instructions
 *
 *  The user also has the option to request pop and breadsticks
 *
 *  The application error checks that valid selections are made when placing the order and
 *  lists error messages listing the problems if there are some.
 *  If there are not errors, the order is placed into a PizzaOrder Object
 *
 *  CST 183 Programming Assignment 8
 *  @author Michael Clinesmith
 *******************************************************************************************/

import javafx.application.Application;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.geometry.*;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;

public class PizzaInterface extends Application
{
    // image objects
    private Image orangeDragon, pizza, pizzaPlaceName;
    private ImageView orangeDragonIV, pizzaIV, pizzaPlaceNameIV;

    private HBox heading, buttonHBox, infoHBox;
    private VBox orderTypeVBox, serverVBox, serverOrderTypeVBox, bottomVBox;

    // left frame objects
    private Label serverLabel, orderTypeLabel;
    private TextField serverName;
    private ComboBox<String> orderType;

    // bottom frame objects
    private Button clearOrderButton, submitOrderButton, exitButton;
    private BorderPane borderPane;
    private TextArea infoTextArea;

    // pizza interface objects
    private GridPane pizzaOrderGrid;
    private Label pizzaSizeLabel, specialtyLabel, addOnLabel, instructionsLabel, spicyLabel;
    private RadioButton smallPizzaRadio, mediumPizzaRadio, largePizzaRadio, xlargePizzaRadio;
```

```
private VBox pizzaSizeVBox, specialtyVBox, addOnVBox, instructionsVBox, spicyVBox;

private ToggleGroup pizzaToggle;
private ComboBox<String> specialty;
private CheckBox extraCheese, extraSauce, garlicCrust, stuffedCrust;
private TextArea specialInstructionsArea;
private Slider spiciness;

// extra options interface objects
private Label popSizeLabel, popTypeLabel, breadstickRadioLabel;
private ToggleGroup popSizeToggle, breadstickToggle;
private CheckBox popCheckBox, breadstickCheckBox;
private ComboBox<String> popTypeComboBox;
private HBox PopTypeHBox;
private VBox extraOptionsVBox, popVBox, breadstickVBox, popOptionVBox, popSizeVBox, popTypeVBox,
                breadstickOptionVBox, breadstickTypeVBox;
private RadioButton smallPopRadio, mediumPopRadio, largePopRadio, xlargePopRadio;
private RadioButton regularBreadsticksRadio, cheesyBreadsticksRadio, frostedBreadsticksRadio;

// main object
private PizzaOrder pizzaObject;

// String to hold error message
private String errorMessage = "No Errors";
/**
 * main method of program, used to launch graphical interface
 *
 * @param args String array - arguments are not used besides being passed to launch method
 */
public static void main(String[] args)
{
    // Launch the application.
    launch(args);
}

/**
 * Method that calls the initializeScene method and creates the scene
 * @param primaryStage Stage object used to create the stage
 */
@Override
public void start(Stage primaryStage)
{
    initializeScene();

    // Set up overall scene

    Scene scene = new Scene(borderPane, 1100, 900);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Pizza Order");
    primaryStage.show();

}

/**
 * Method calls other methods to design the user interface
 */
public void initializeScene()
{
    loadImages();
    prepareHeader();
```

```
    prepareOrderTypeBox();
    prepareOrderGrid();
    prepareExtraOptions();
    prepareBottomBox();

    borderPane = new BorderPane();
    displayMainMenu();
}

/**
 * Method to create the borderPane for the interface
 */
public void displayMainMenu()
{
    // the first two lines of code are elements of the design process which changed during development
    // and could be moved or changed later if necessary
    buttonHBox.getChildren().clear();
    buttonHBox.getChildren().addAll(submitOrderButton, clearOrderButton, exitButton );

    borderPane.setTop(heading);
    borderPane.setLeft(serverOrderTypeVBox);
    borderPane.setRight(extraOptionsVBox);
    borderPane.setBottom(bottomVBox);
    borderPane.setCenter(pizzaOrderGrid);

}

/**
 * Method to upload and set up images for interface
 */
public void loadImages()
{
    orangeDragon = new Image("file:Cute orange dragon.jpg");
    pizza = new Image("file:pizza.jpg");
    pizzaPlaceName = new Image("file:AirysPizzaPalace.png");

    orangeDragonIV = new ImageView(orangeDragon);
    pizzaIV = new ImageView(pizza);
    pizzaPlaceNameIV = new ImageView(pizzaPlaceName);

    orangeDragonIV.setFitHeight(100);
    orangeDragonIV.setPreserveRatio(true);

    pizzaIV.setFitHeight(100);
    pizzaIV.setPreserveRatio(true);

    pizzaPlaceNameIV.setFitHeight(100);
    pizzaPlaceNameIV.setPreserveRatio(true);

}

/**
 * Method to create the interface header
 */
public void prepareHeader()
{
    heading = new HBox(pizzaPlaceNameIV, pizzaIV, orangeDragonIV);
    heading.setStyle("-fx-background-color: white");
}
```

```
/**
 * Method to setup the server name and delivery type on the left side of the interface
 */
public void prepareOrderTypeBox()
{
    serverLabel = new Label(("Server Name"));
    serverName = new TextField();
    serverVBox = new VBox(serverLabel, serverName);
    serverVBox.setAlignment(Pos.TOP_CENTER);

    orderTypeLabel = new Label("Type of Delivery");
    orderType = new ComboBox<String>();
    orderType.getItems().addAll("Dine In", "Carry Out", "Delivery");
    orderTypeVBox = new VBox(orderTypeLabel, orderType);
    orderTypeVBox.setAlignment(Pos.TOP_CENTER);
    serverOrderTypeVBox = new VBox(100, serverVBox, orderTypeVBox);
    serverOrderTypeVBox.setAlignment(Pos.TOP_CENTER);
    serverOrderTypeVBox.setPadding(new Insets(30));
    serverOrderTypeVBox.setStyle("-fx-background-color: yellow");
}

/**
 * Method to prepare the buttons at the bottom of the screen
 */
public void prepareButtons()
{
    submitOrderButton = new Button("Submit Order");
    clearOrderButton = new Button("Clear Order");
    exitButton = new Button( "Exit Program");

    clearOrderButton.setOnAction(new ButtonClickHandler());
    submitOrderButton.setOnAction(new ButtonClickHandler());
    exitButton.setOnAction(new ButtonClickHandler());

    // buttonHBox = new HBox( 100, beginOrderButton, reviewOrderButton, exitButton );
    buttonHBox = new HBox( 100 );
    buttonHBox.setAlignment(Pos.CENTER);
    buttonHBox.setPadding(new Insets(30));

}

/**
 * Method that calls other methods to design the center order grid
 */
public void prepareOrderGrid()
{
    preparePizzaSizeOption();
    preparePizzaSpecialityOption();
    preparePizzaAddOnIngredients();
    prepareSpecialInstructions();
    prepareSpicyOption();
    createGridPane();
}

/**
 * Method to design the pizza size option
 */
public void preparePizzaSizeOption()
{
    pizzaSizeLabel = new Label("Pizza Size:");
```

```
    smallPizzaRadio = new RadioButton("Small: 6\"");
    mediumPizzaRadio = new RadioButton("Medium: 10\"");
    largePizzaRadio = new RadioButton("Large: 12\"");
    xlargePizzaRadio = new RadioButton("X-large: 14\"");
    pizzaToggle = new ToggleGroup();
    smallPizzaRadio.setToggleGroup(pizzaToggle);
    mediumPizzaRadio.setToggleGroup(pizzaToggle);
    largePizzaRadio.setToggleGroup(pizzaToggle);
    xlargePizzaRadio.setToggleGroup(pizzaToggle);

    pizzaSizeVBox = new VBox(10, pizzaSizeLabel, smallPizzaRadio, mediumPizzaRadio, largePizzaRadio, xlargePizzaRadio);
    pizzaSizeVBox.setPadding(new Insets(30));

}

/**
 * Method to design the pizza specialty choice option
 */
public void preparePizzaSpecialityOption()
{

    specialty = new ComboBox<String>();
    specialtyLabel = new Label("Pizza Type:");
    specialty.getItems().addAll("Supreme", "Meat", "Cheese", "Veggie", "Fruit", "Chocolate");
    specialtyVBox = new VBox(10, specialtyLabel, specialty);
    specialtyVBox.setAlignment(Pos.TOP_CENTER);
    specialtyVBox.setPadding(new Insets(30));

}

/**
 * Method to design the extra ingredient options
 */
public void preparePizzaAddOnIngredients()
{
    addOnLabel = new Label("Add-On Ingredients:");
    extraCheese = new CheckBox("Extra Cheese");
    extraSauce = new CheckBox("Extra Sauce");
    garlicCrust = new CheckBox("Garlic Crust");
    stuffedCrust = new CheckBox( "Stuffed Crust");
    addOnVBox = new VBox(10, addOnLabel, extraCheese, extraSauce, garlicCrust, stuffedCrust);
    addOnVBox.setPadding(new Insets(30));
}

/**
 * Method to design the special instructions box
 */
public void prepareSpecialInstructions()
{
    instructionsLabel = new Label("Special Instructions:");
    specialInstructionsArea = new TextArea();
    specialInstructionsArea.setPrefColumnCount(15);
    specialInstructionsArea.setPrefRowCount(8);
    specialInstructionsArea.setWrapText(true);

    instructionsVBox = new VBox(10, instructionsLabel,specialInstructionsArea );
    instructionsVBox.setPadding(new Insets(30));
}

/**
```

Maybe an array declared up-top,

```
 * Method to design the spicy option lable
 */
public void prepareSpicyOption()
{
    spicyLabel = new Label("Sauce Spiciness Level:");
    spiciness = new Slider(0, 20, 0);
    spiciness.setOrientation(Orientation.VERTICAL);
    spiciness.setShowTickMarks(true);
    spiciness.setShowTickLabels(true);
    spiciness.setMajorTickUnit(5);
    spiciness.setMinorTickCount(1);
    spiciness.setSnapToTicks(true);

    spicyVBox = new VBox(10, spicyLabel, spiciness);
    spicyVBox.setPadding(new Insets(30));
}

/**
 * Method that designs the center gridpane and incorperates the other designed elements into it
 */
public void createGridPane()
{
    pizzaOrderGrid = new GridPane();
    pizzaOrderGrid.add(pizzaSizeVBox, 0, 0);
    pizzaOrderGrid.add(specialtyVBox, 1, 0);
    pizzaOrderGrid.add(addOnVBox, 2, 0);
    pizzaOrderGrid.add(instructionsVBox, 0, 1);
    pizzaOrderGrid.add(spicyVBox, 1, 1);
    pizzaOrderGrid.setAlignment(Pos.CENTER);
    pizzaOrderGrid.setStyle("-fx-background-color: orange");
}

/**
 * Method that designs the box that includes the buttons and informational area at the bottom of the interface
 */
public void prepareBottomBox()
{
    prepareButtons();
    infoTextArea = new TextArea("Satisfy your dragon sized hunger at Airy's Pizza Palace!");
    infoTextArea.setPrefColumnCount(40);
    infoTextArea.setPrefRowCount(5);
    infoTextArea.setEditable(false);
    infoHBox = new HBox(infoTextArea);
    infoHBox.setAlignment(Pos.CENTER);
    infoHBox.setPadding(new Insets(30));
    bottomVBox = new VBox(20, infoHBox, buttonHBox );
    bottomVBox.setStyle("-fx-background-color: red");
}

/**
 * Method that designs the extra options on the right side of the screen
 */
public void prepareExtraOptions()
{
    preparePopOption();
    prepareBreadstickOption();

    extraOptionsVBox = new VBox(popVBox, breadstickVBox);
    extraOptionsVBox.setPadding(new Insets(30));
    extraOptionsVBox.setStyle("-fx-background-color: yellow");
```

```
}

/**
 * Method that designs the pop options, some do not show initially until the user selects the option
 */
public void preparePopOption()
{

    // frame for Check Box for pop
    popCheckBox = new CheckBox("Include Pop?");
    popCheckBox.setOnAction(new CheckBoxClickHandler());

    popOptionVBox = new VBox(10, popCheckBox);

    // frame for RadioButtons for size
    popSizeLabel = new Label("Pop Size");
    smallPopRadio = new RadioButton("Small: 16 oz");
    mediumPopRadio = new RadioButton("Medium: 24 oz");
    largePopRadio = new RadioButton("Large: 32 oz");
    xlargePopRadio = new RadioButton("X-Large 42 oz");
    popSizeToggle = new ToggleGroup();
    smallPopRadio.setToggleGroup(popSizeToggle);
    mediumPopRadio.setToggleGroup(popSizeToggle);
    largePopRadio.setToggleGroup(popSizeToggle);
    xlargePopRadio.setToggleGroup(popSizeToggle);

    popSizeVBox = new VBox(10, smallPopRadio, mediumPopRadio, largePopRadio, xlargePopRadio);

    // frame for flavor choice
    popTypeLabel = new Label("Pop Flavor:");
    popTypeComboBox = new ComboBox<String>();
    popTypeComboBox.getItems().addAll("Pepsi", "Coke", "Red Pop", "Root Beer", "Vernors");

    popTypeVBox = new VBox(10, popTypeLabel, popTypeComboBox);

    //  put frames together, only show one option until user checks
    popVBox = new VBox(10, popCheckBox);
    popVBox.setPadding(new Insets(30));

}

/**
 * Method that designs the breadstick options, some do not show initially until the user selects the option
 */
public void prepareBreadstickOption()
{

    // frame for breadstick option box
    breadstickCheckBox = new CheckBox("Include Breadsticks?");
    breadstickCheckBox.setOnAction(new CheckBoxClickHandler());
    breadstickOptionVBox = new VBox (10, breadstickCheckBox);

    // frame for breadstick flavors radiobuttons
    breadstickRadioLabel = new Label("Breadstick Type");
    regularBreadsticksRadio = new RadioButton("Regular");
    cheesyBreadsticksRadio = new RadioButton( "Cheesy");
    frostedBreadsticksRadio = new RadioButton("Frosted");
    breadstickToggle = new ToggleGroup();
    regularBreadsticksRadio.setToggleGroup(breadstickToggle);
    cheesyBreadsticksRadio.setToggleGroup(breadstickToggle);
```

```
    frostedBreadsticksRadio.setToggleGroup(breadstickToggle);
    breadstickTypeVBox = new VBox(10, breadstickRadioLabel, regularBreadsticksRadio,
                                          cheesyBreadsticksRadio, frostedBreadsticksRadio);

    // connect frames for breadstick option choice, only show one option until user checks
    breadstickVBox = new VBox(10, breadstickOptionVBox);
    breadstickVBox.setPadding(new Insets(30));
}

/**
 * Method that makes the pop options appear on the interface
 */
public void showPopOptions()
{
    popVBox.getChildren().clear();
    popVBox.getChildren().addAll( popCheckBox, popSizeVBox, popTypeVBox);

}

/**
 * Method that makes the pop options disappear on the interface
 */
public void hidePopOptions()
{
    popVBox.getChildren().clear();
    popVBox.getChildren().addAll( popCheckBox);

}

/**
 * Method that makes the breadstick options appear on the interface
 */
public void showBreadstickOptions()
{
    breadstickVBox.getChildren().clear();
    breadstickVBox.getChildren().addAll( breadstickOptionVBox, breadstickTypeVBox);

}

/**
 * Method that makes the breadstick options disappear on the interface
 */
public void hideBreadstickOptions()
{
    breadstickVBox.getChildren().clear();
    breadstickVBox.getChildren().addAll( breadstickOptionVBox);

}

/**
 * Method that returns the selection make regarding the pizza size choice
 * @return String representing the pizza size choice selected, or "None Selected" if none selected
 */
public String getPizzaSizeChoice()
{
    String sizeChoice="None Selected";
    if (smallPizzaRadio.isSelected())
    {
        sizeChoice = smallPizzaRadio.getText();
    }
```

```
    else if(mediumPizzaRadio.isSelected())
    {
        sizeChoice = mediumPizzaRadio.getText();
    }
    else if(largePizzaRadio.isSelected())
    {
        sizeChoice = largePizzaRadio.getText();
    }
    else if(xlargePizzaRadio.isSelected())
    {
        sizeChoice = xlargePizzaRadio.getText();
    }

    return sizeChoice;
}

/**
 * Method that returns a value based on if the options selected in the interface are valid
 * The method also creates an error message based on invalid selections
 * @return boolean true if selections are valid, false if not
 */
public boolean isSelectionsValid()
{
    boolean isValid = true;
    errorMessage = "";

    if(specialty.getValue()==null || specialty.getValue().length()==0)
    {
        isValid = false;
        errorMessage += "Specialty pizza Style not chosen.\n";
    }
    if(serverName.getText().length()==0)
    {
        isValid = false;
        errorMessage += "Server name not entered.\n";
    }
    if(orderType.getValue() == null || orderType.getValue().length()==0)
    {
        isValid = false;
        errorMessage += "Delivery method not entered.\n";
    }
    if(getPizzaSizeChoice().equals("None Selected"))
    {
        isValid = false;
        errorMessage += "Pizza size not selected.\n";
    }
    if (popCheckBox.isSelected())
    {
        if (!smallPopRadio.isSelected() && !mediumPopRadio.isSelected() && !largePopRadio.isSelected()
            && !xlargePopRadio.isSelected())  // no pop size selected
        {
            isValid = false;
            errorMessage += "Pop size not selected.\n";
        }
        if (popTypeComboBox.getValue() == null || popTypeComboBox.getValue().length()==0)
        {
            isValid = false;
            errorMessage += "Pop flavor not selected.\n";
        }
```

```
    }
    if (breadstickCheckBox.isSelected())
    {
        if(!regularBreadsticksRadio.isSelected() && !cheesyBreadsticksRadio.isSelected()
                    && !frostedBreadsticksRadio.isSelected())       // no breadstick type selected
        {
            isValid = false;
            errorMessage += "No breadstick type selected.\n";
        }
    }

    return isValid;
}

/**
 * Class ButtonClickHandler handles the button click events
 */
class ButtonClickHandler implements EventHandler<ActionEvent>
{
    /**
     * This method handles button click events
     *
     * If the submit order button is clicked, the selections are checked to see if the are valid
     * if the selections are valid, a pizza object is created and the order displayed
     * if some selections are invalid, an error message is displayed to the user
     *
     * If the clear button is clicked, the settings in the interface are cleared
     *
     * If the quit button is clicked, the program ends
     *
     * @param event ActionEvent object that contains data about a button click event
     */
    @Override
    public void handle(ActionEvent event)
    {

        if (event.getSource() == submitOrderButton)             // calculate based on data
        {
            // check if valid selections made
            if(isSelectionsValid())
            {
                // create pizzaObject
                pizzaObject = new PizzaOrder(specialty.getValue(),
                        serverName.getText(),
                        orderType.getValue(),
                        getPizzaSizeChoice(),
                        extraCheese.isSelected(),
                        extraSauce.isSelected(),
                        garlicCrust.isSelected(),
                        stuffedCrust.isSelected(),
                        spiciness.getValue(),
                        specialInstructionsArea.getText());

                infoTextArea.setText(pizzaObject.toString()+"Order confirmed.");
            }
            else
            {
                infoTextArea.setText(errorMessage +"Order not placed.");
            }
```

```
        }
        if (event.getSource() == clearOrderButton)
        {
            serverName.setText("");
            orderType.setValue("");
            // deselect pizza size radio buttons
            smallPizzaRadio.setSelected(true);
            smallPizzaRadio.setSelected(false);

            specialty.setValue("");
            extraCheese.setSelected(false);
            extraSauce.setSelected(false);
            garlicCrust.setSelected(false);
            stuffedCrust.setSelected(false);
            specialInstructionsArea.setText("");
            spiciness.setValue(0.0);
            popCheckBox.setSelected(false);
            // deselect pop size radio buttons
            smallPopRadio.setSelected(true);
            smallPopRadio.setSelected(false);

            popTypeComboBox.setValue("");
            breadstickCheckBox.setSelected(false);
            // deselect breadstick type options
            regularBreadsticksRadio.setSelected(true);
            regularBreadsticksRadio.setSelected(false);

            // remove extra pop and breadstick options if exist
            hidePopOptions();
            hideBreadstickOptions();

            infoTextArea.setText("Menu options have been cleared.");

        }
        if (event.getSource() == exitButton)
        {
            System.exit(0);
        }

    }
}

/**
 * class CheckBoxClickHandler addresses the extra checkbox selections
 */
class CheckBoxClickHandler implements EventHandler<ActionEvent>
{
    /**
     * This method handles the extra checkbox click events
     *
     * If the include pop option is clicked or unclicked, the interface is updated to increase or decrease
     * the pop options
     *
     * If the include breadsticks option is clicked or unclicked, the interface is updated to increase or
     * decrease the breadsticks options
     *
     * @param event ActionEvent object that contains data about a checkbox click event
     */
    @Override
    public void handle(ActionEvent event)
```

```
        {

            if (event.getSource() == popCheckBox)             // calculate based on data
            {
                if (popCheckBox.isSelected())
                {
                    showPopOptions();
                }
                else
                {
                    hidePopOptions();
                }

            }
            if (event.getSource() == breadstickCheckBox)
            {
                if (breadstickCheckBox.isSelected())
                {
                    showBreadstickOptions();
                }
                else
                {
                    hideBreadstickOptions();
                }
            }

        }
    }

}
```