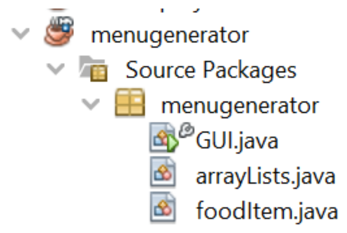
Criterion C: Development

All classes


menugenerator
Source Packages
menugenerator
GUI.java
arrayLists.java
foodItem.java

Libraries Imported

```
import java.util.ArrayList;
import javax.swing.*;
import java.io.*;
import java.nio.file.*;
import javax.swing.filechooser.FileNameExtensionFilter;

import java.io.Serializable;
```

List of techniques used

1. The GUI is arranged using layout managers such as GridLayout
2. Exceptions are handled using try-catch blocks
3. Polymorphism is used to allow an Object variable to access the methods and fields of a specific class.
4. Getters and setters are defined to access and modify fields.
5. ArrayLists are used to store and manipulate menu items and categories
6. ObjectOutputStream and PrintWriter are used to write the menu data to a file in the fileSaveActionPerformed method
7. An ObjectInputStream is used to read an object from a file in the fileOpenActionPerformed method.

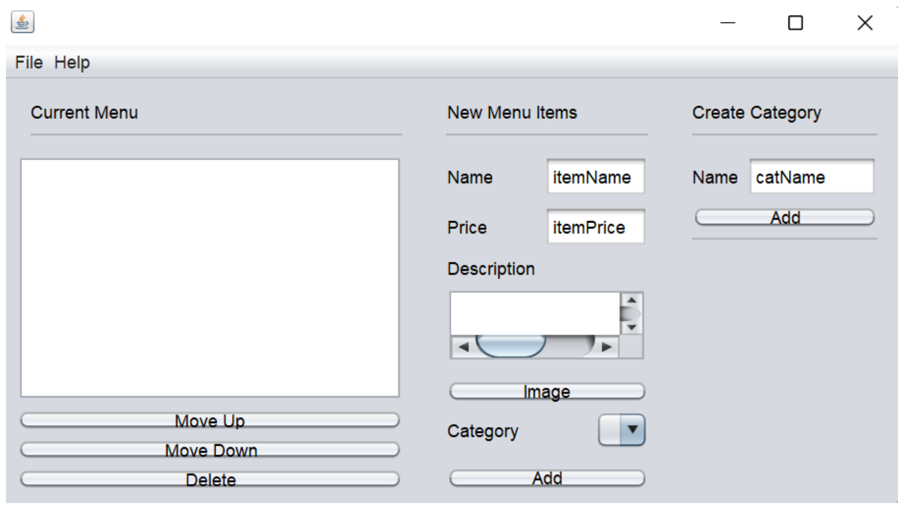
Software Tools

NetBeans was the chosen IDE, which helped me create and customize the GUI through its drag-and-drop interface. The debugger in NetBeans was also used to identify and fix errors in the code, allowing me to step through the code and examine the values of variables and the flow of execution at different points.

Usability of Interface

- The interface includes clear labels and instructions, such as the “Move Up" label, helping users understand the purpose of each element.
- The interface provides clear feedback when an action has been completed, such as the "menuList" list being updated when a new menu item is added. This helps users understand the actions they have taken.
- The interface includes checks to prevent errors, such as the check for an existing category name before a new category is added. If so, a warning message appears.
- The interface allows users to easily add and organize menu items, with the minimum number of button presses.

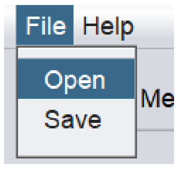
Main Window

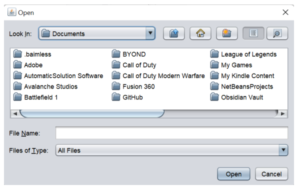


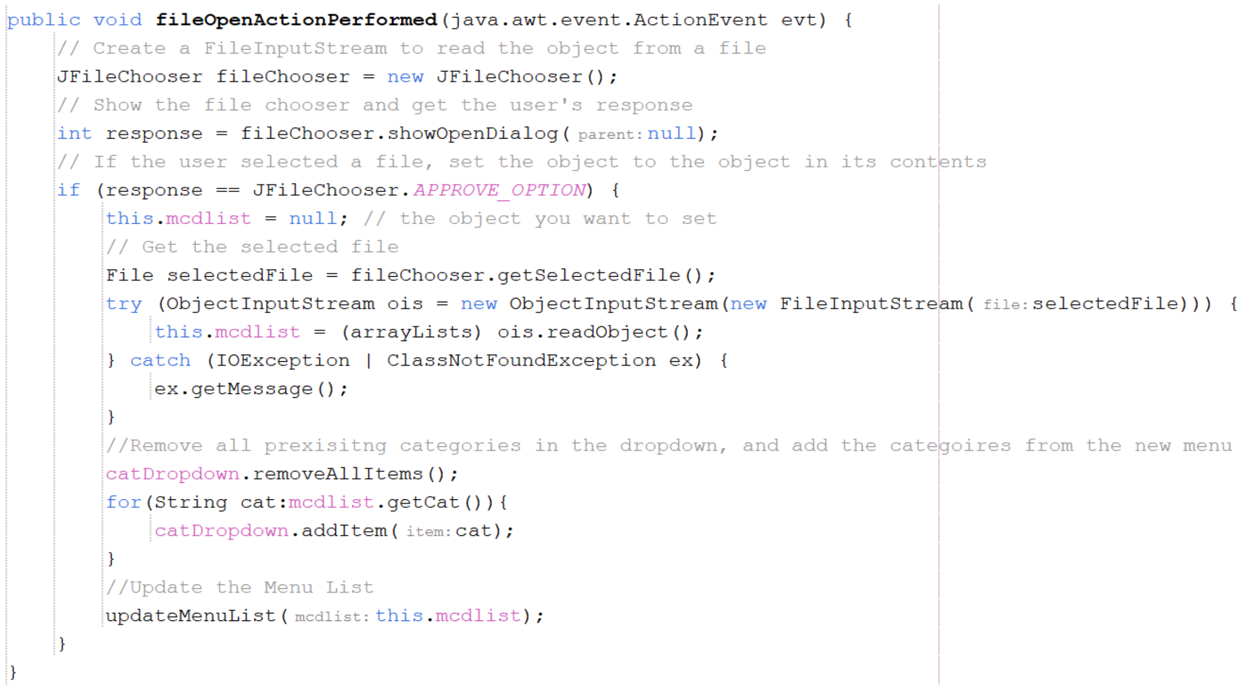
```
jDialog1 = new javax.swing.JDialog();
jDialog2 = new javax.swing.JDialog();
jDialog3 = new javax.swing.JDialog();
jDialog4 = new javax.swing.JDialog();
jPanel1 = new javax.swing.JPanel();
jPanel2 = new javax.swing.JPanel();
currentLabel = new javax.swing.JLabel();
jSeparator3 = new javax.swing.JSeparator();
jScrollPane1 = new javax.swing.JScrollPane();
menuList = new javax.swing.JList<>();
moveUp = new javax.swing.JButton();
moveDown = new javax.swing.JButton();
menuDelete = new javax.swing.JButton();
jPanel3 = new javax.swing.JPanel();
categoryLabel = new javax.swing.JLabel();
jSeparator1 = new javax.swing.JSeparator();
catName = new javax.swing.JTextField();
cNameLabel = new javax.swing.JLabel();
addCategory = new javax.swing.JButton();
jSeparator4 = new javax.swing.JSeparator();
jPanel4 = new javax.swing.JPanel();
newLabel = new javax.swing.JLabel();
jSeparator2 = new javax.swing.JSeparator();
menuName = new javax.swing.JTextField();
mNameLabel = new javax.swing.JLabel();
mPriceLabel = new javax.swing.JLabel();
menuPrice = new javax.swing.JTextField();
mDescLabel = new javax.swing.JLabel();
menuImage = new javax.swing.JButton();
catDropdown = new javax.swing.JComboBox<>();
mCategoryLabel = new javax.swing.JLabel();
addMenu = new javax.swing.JButton();
jScrollPane3 = new javax.swing.JScrollPane();
menuDescription = new javax.swing.JTextArea();
jMenuBar1 = new javax.swing.JMenuBar();
file = new javax.swing.JMenu();
fileOpen = new javax.swing.JMenuItem();
fileSave = new javax.swing.JMenuItem();
fileExport = new javax.swing.JMenuItem();
help = new javax.swing.JMenu();
helpInstruction = new javax.swing.JMenuItem();
```

The code creates and arranges GUI components, such as labels and text fields, using JGridLayout.

File Selection



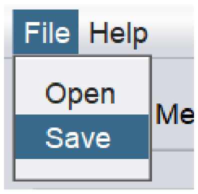


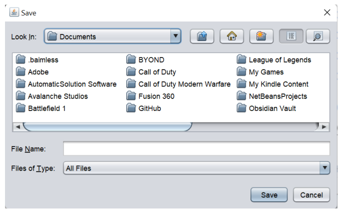


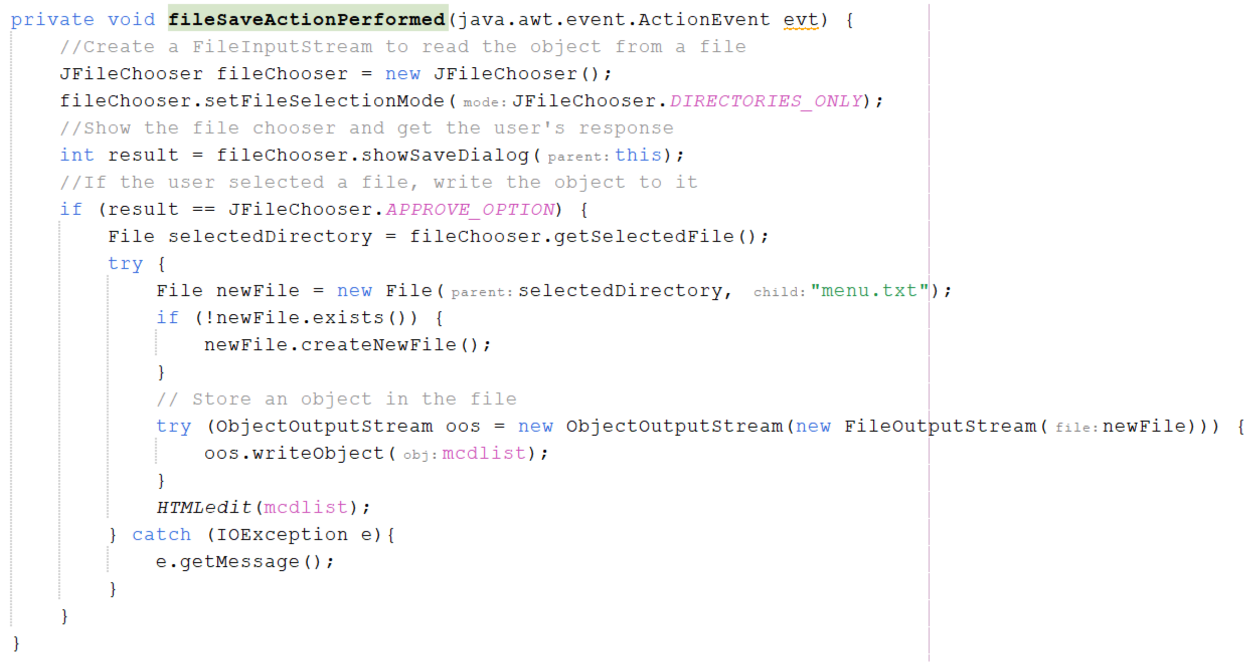
```java
public void fileOpenActionPerformed(java.awt.event.ActionEvent evt) {
    // Create a FileInputStream to read the object from a file
    JFileChooser fileChooser = new JFileChooser();
    // Show the file chooser and get the user's response
    int response = fileChooser.showOpenDialog( parent: null);
    // If the user selected a file, set the object to the object in its contents
    if (response == JFileChooser.APPROVE_OPTION) {
        this.mcdlist = null; // the object you want to set
        // Get the selected file
        File selectedFile = fileChooser.getSelectedFile();
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream( file: selectedFile))) {
            this.mcdlist = (arrayLists) ois.readObject();
        } catch (IOException | ClassNotFoundException ex) {
            ex.getMessage();
        }
        //Remove all prexisitng categories in the dropdown, and add the categoires from the new menu
        catDropdown.removeAllItems();
        for(String cat:mcdlist.getCat()){
            catDropdown.addItem( item: cat);
        }
        //Update the Menu List
        updateMenuList( mcdlist: this.mcdlist);
    }
}
```

The code reads an object from a file, updates the dropdown menu and menu list display with the new data, and stores it in an "arrayLists" object.

This method could be modified to display an error message to the user in an exception.





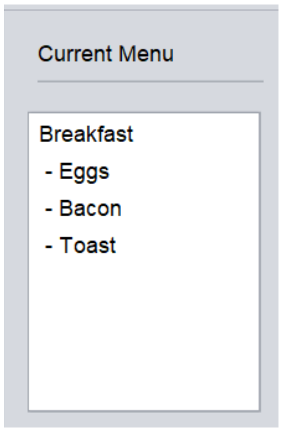


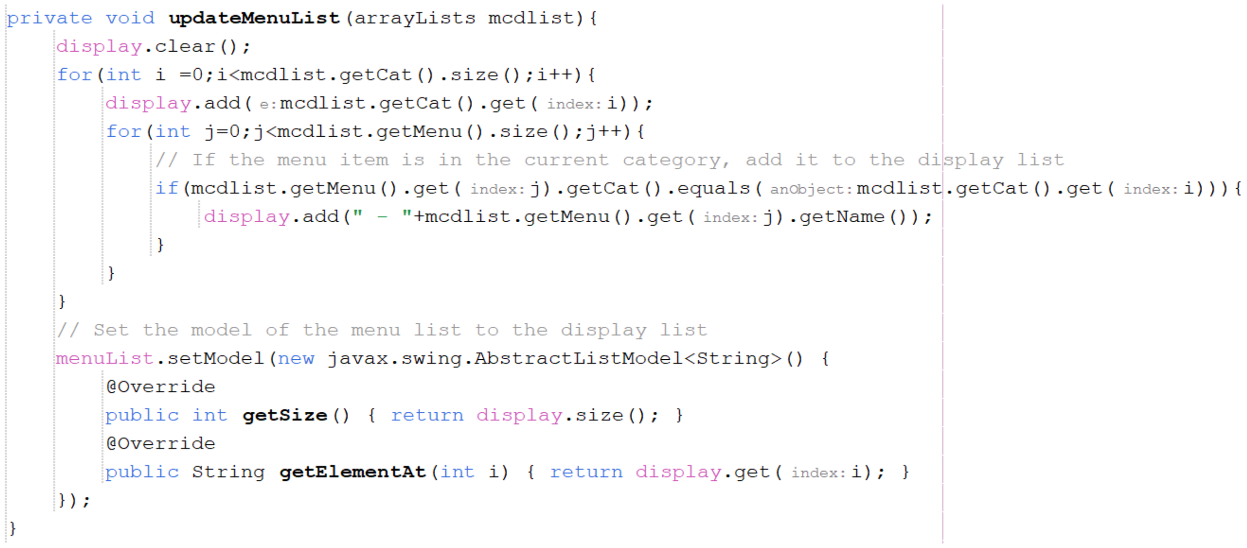
```
private void fileSaveActionPerformed(java.awt.event.ActionEvent evt) {
    //Create a FileInputStream to read the object from a file
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode( mode: JFileChooser.DIRECTORIES_ONLY);
    //Show the file chooser and get the user's response
    int result = fileChooser.showSaveDialog( parent: this);
    //If the user selected a file, write the object to it
    if (result == JFileChooser.APPROVE_OPTION) {
        File selectedDirectory = fileChooser.getSelectedFile();
        try {
            File newFile = new File( parent: selectedDirectory,  child: "menu.txt");
            if (!newFile.exists()) {
                newFile.createNewFile();
            }
            // Store an object in the file
            try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( file: newFile))) {
                oos.writeObject( obj: mcdlist);
            }
            HTMLedit(mcdlist);
        } catch (IOException e){
            e.getMessage();
        }
    }
}
```

Code uses JFileChooser, allowing the user to select a directory, creating a new File in that directory, and writing the object to the file.

It could be modified to prompt the user to confirm overwriting an existing file and handle more exceptions, such as FileNotFoundException.

Ordering Menu Items

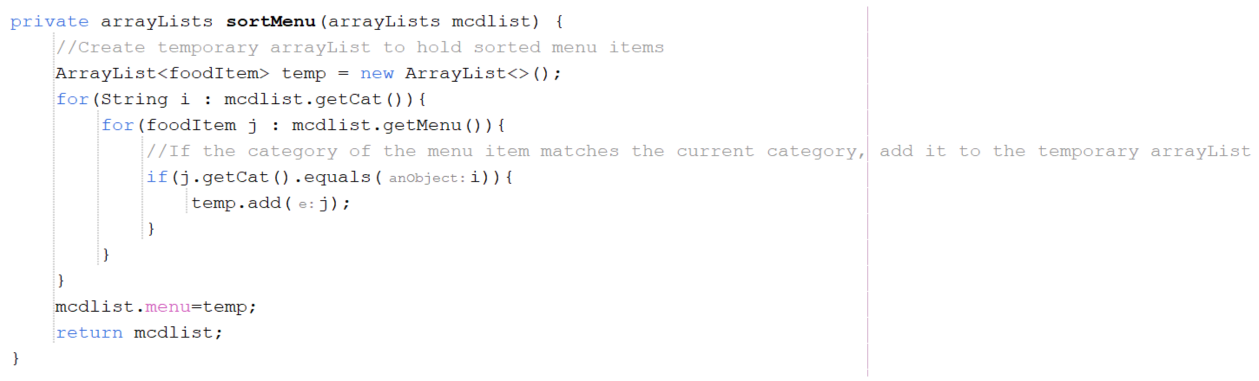




```java
private void updateMenuList(arrayLists mcdlist){
    display.clear();
    for(int i =0;i<mcdlist.getCat().size();i++){
        display.add( e:mcdlist.getCat().get( index: i));
        for(int j=0;j<mcdlist.getMenu().size();j++){
            // If the menu item is in the current category, add it to the display list
            if(mcdlist.getMenu().get( index: j).getCat().equals( anObject: mcdlist.getCat().get( index: i))){
                display.add(" - "+mcdlist.getMenu().get( index: j).getName());
            }
        }
    }
    // Set the model of the menu list to the display list
    menuList.setModel(new javax.swing.AbstractListModel<String>() {
        @Override
        public int getSize() { return display.size(); }
        @Override
        public String getElementAt(int i) { return display.get( index: i); }
    });
}
```

Code updates the menuList component by iterating through the categories in an ArrayList, adding each one to a display list, and then setting the menuList's model to the display list. It uses ArrayLists and foodItem objects to store the menu data.

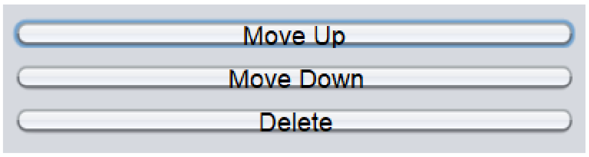
Code could be modified or expanded to display more information about each menu item such as price.

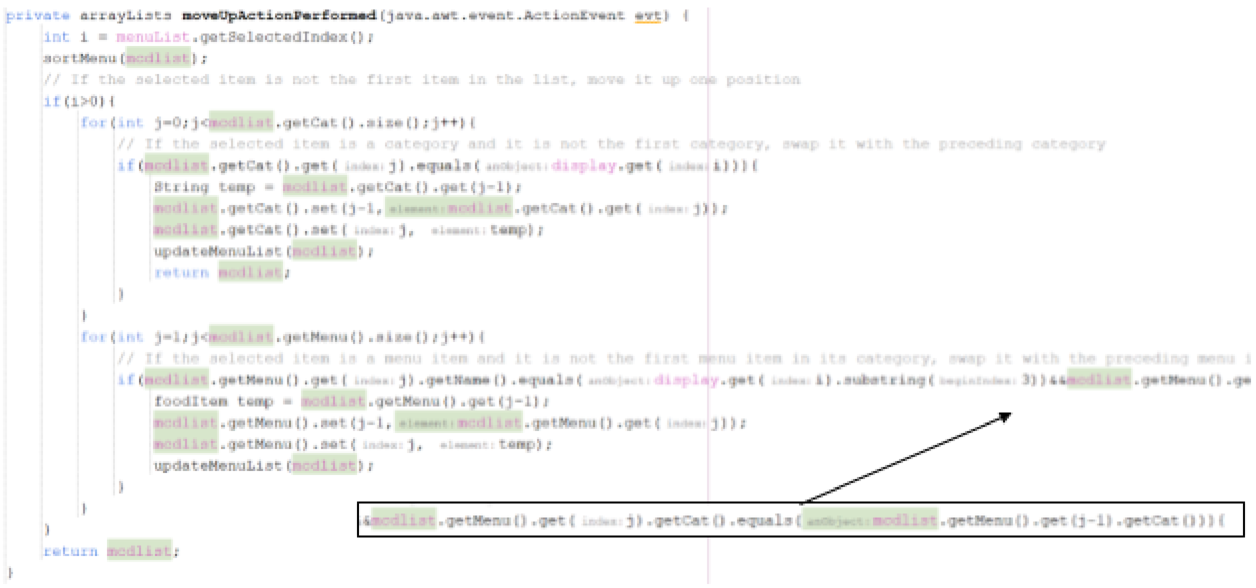


```
private arrayLists sortMenu(arrayLists mcdlist) {
    //Create temporary arrayList to hold sorted menu items
    ArrayList<foodItem> temp = new ArrayList<>();
    for(String i : mcdlist.getCat()){
        for(foodItem j : mcdlist.getMenu()){
            //If the category of the menu item matches the current category, add it to the temporary arrayList
            if(j.getCat().equals( anObject: i)){
                temp.add( e: j);
            }
        }
    }
    mcdlist.menu=temp;
    return mcdlist;
}
```

The method "sortMenu" takes in an "arrayLists" object and returns the same object with the "menu" field sorted by category. It uses an ArrayList to store the sorted menu items and iterates through the "cat" and "menu" fields of the input "arrayLists" object to do so.

This code can be modified to sort the menu items in a different order (e.g. by price, by name, etc.) by modifying the comparison used to add items to the temporary list.



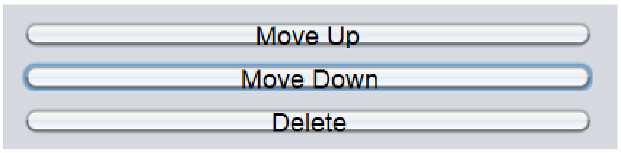


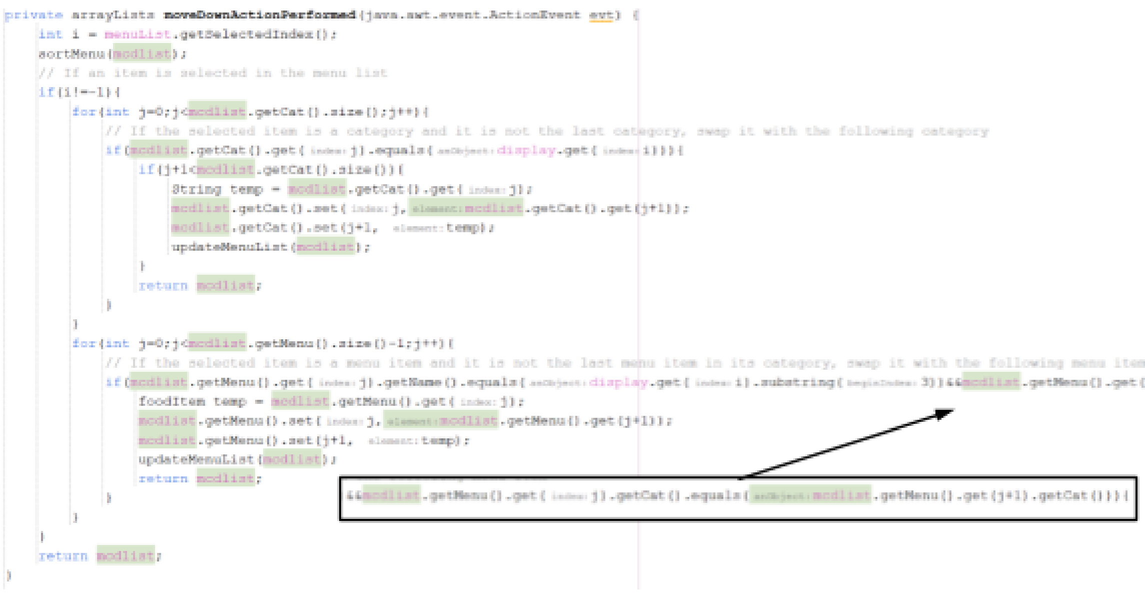
```
private arrayLists moveUpActionPerformed(java.awt.event.ActionEvent evt) {
    int i = menuList.getSelectedIndex();
    sortMenu(modlist);
    // If the selected item is not the first item in the list, move it up one position
    if(i>0){
        for(int j=0;j<modlist.getCat().size();j++){
            // If the selected item is a category and it is not the first category, swap it with the preceding category
            if(modlist.getCat().get( index: j).equals( anObject: display.get( index: i))){
                String temp = modlist.getCat().get(j-1);
                modlist.getCat().set(j-1, element: modlist.getCat().get( index: j));
                modlist.getCat().set( index: j,  element: temp);
                updateMenuList(modlist);
                return modlist;
            }
        }
        for(int j=1;j<modlist.getMenu().size();j++){
            // If the selected item is a menu item and it is not the first menu item in its category, swap it with the preceding menu i
            if(modlist.getMenu().get( index: j).getName().equals( anObject: display.get( index: i).substring( beginIndex: 3))&&modlist.getMenu().ge
                foodItem temp = modlist.getMenu().get(j-1);
                modlist.getMenu().set(j-1, element: modlist.getMenu().get( index: j));
                modlist.getMenu().set( index: j,  element: temp);
                updateMenuList(modlist);
            }
        }
    }
    return modlist;
}
```

```
&modlist.getMenu().get( index: j).getCat().equals( anObject: modlist.getMenu().get(j-1).getCat())){
```

It sorts the menu, then checks if the selected item is not the first in the list. If it's not, the method checks if the item is a category or a menu item, then swaps it with the preceding item of the same type.

It could be modified to handle the case where the selected item is the first item in the list, and it moves the item to the bottom instead .

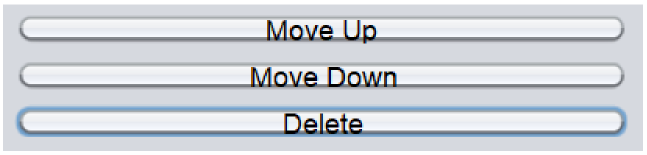


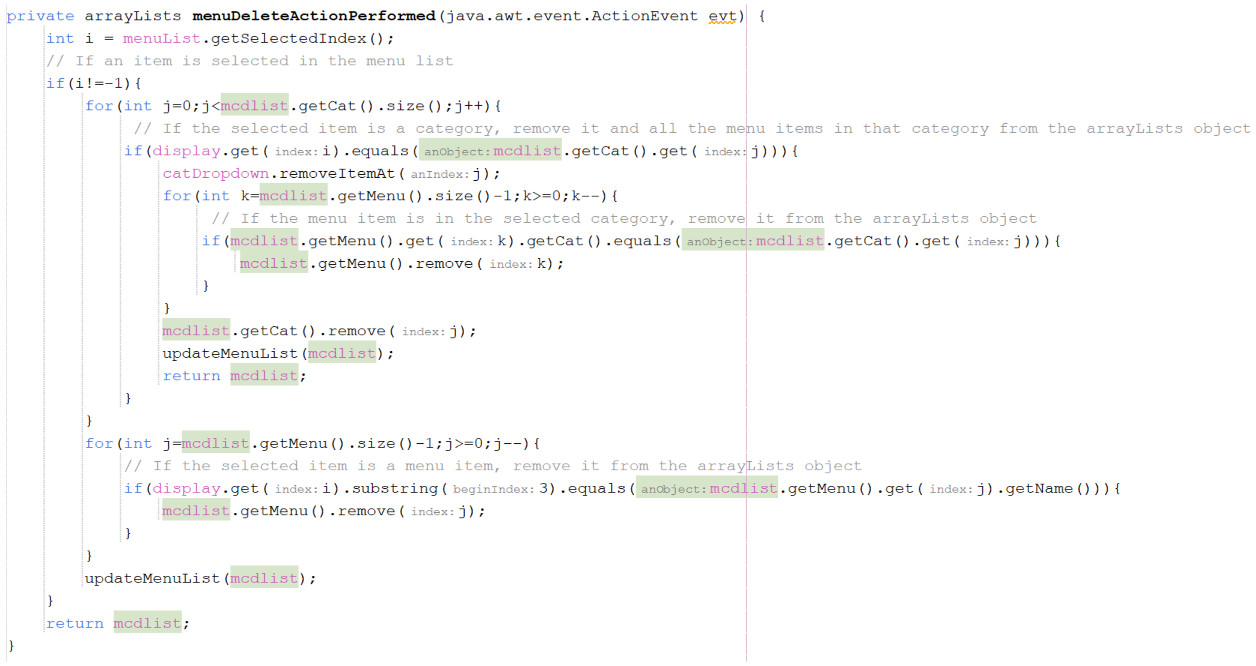


```
private arrayLists moveDownActionPerformed(java.awt.event.ActionEvent evt) {
    int i = menuList.getSelectedIndex();
    sortMenu(modlist);
    // If an item is selected in the menu list
    if(i!=-1){
        for(int j=0;j<modlist.getCat().size();j++){
            // If the selected item is a category and it is not the last category, swap it with the following category
            if(modlist.getCat().get( index: j).equals( anObject: display.get( index: i))){
                if(j+1<modlist.getCat().size()){
                    String temp = modlist.getCat().get( index: j);
                    modlist.getCat().set( index: j, element: modlist.getCat().get(j+1));
                    modlist.getCat().set(j+1,  element: temp);
                    updateMenuList(modlist);
                }
                return modlist;
            }
        }
        for(int j=0;j<modlist.getMenu().size()-1;j++){
            // If the selected item is a menu item and it is not the last menu item in its category, swap it with the following menu item
            if(modlist.getMenu().get( index: j).getName().equals( anObject: display.get( index: i).substring( beginIndex: 3))&&modlist.getMenu().get( i
                foodItem temp = modlist.getMenu().get( index: j);
                modlist.getMenu().set( index: j, element: modlist.getMenu().get(j+1));
                modlist.getMenu().set(j+1,  element: temp);
                updateMenuList(modlist);
                return modlist;
            }
        }
    }
    return modlist;
}
```

```
&&modlist.getMenu().get( index: j).getCat().equals( anObject: modlist.getMenu().get(j+1).getCat())){
```

It sorts the menu and moves a selected item down in the list if it is not already at the bottom. It handles both categories and menu items, and updates the menu list display after moving an item.

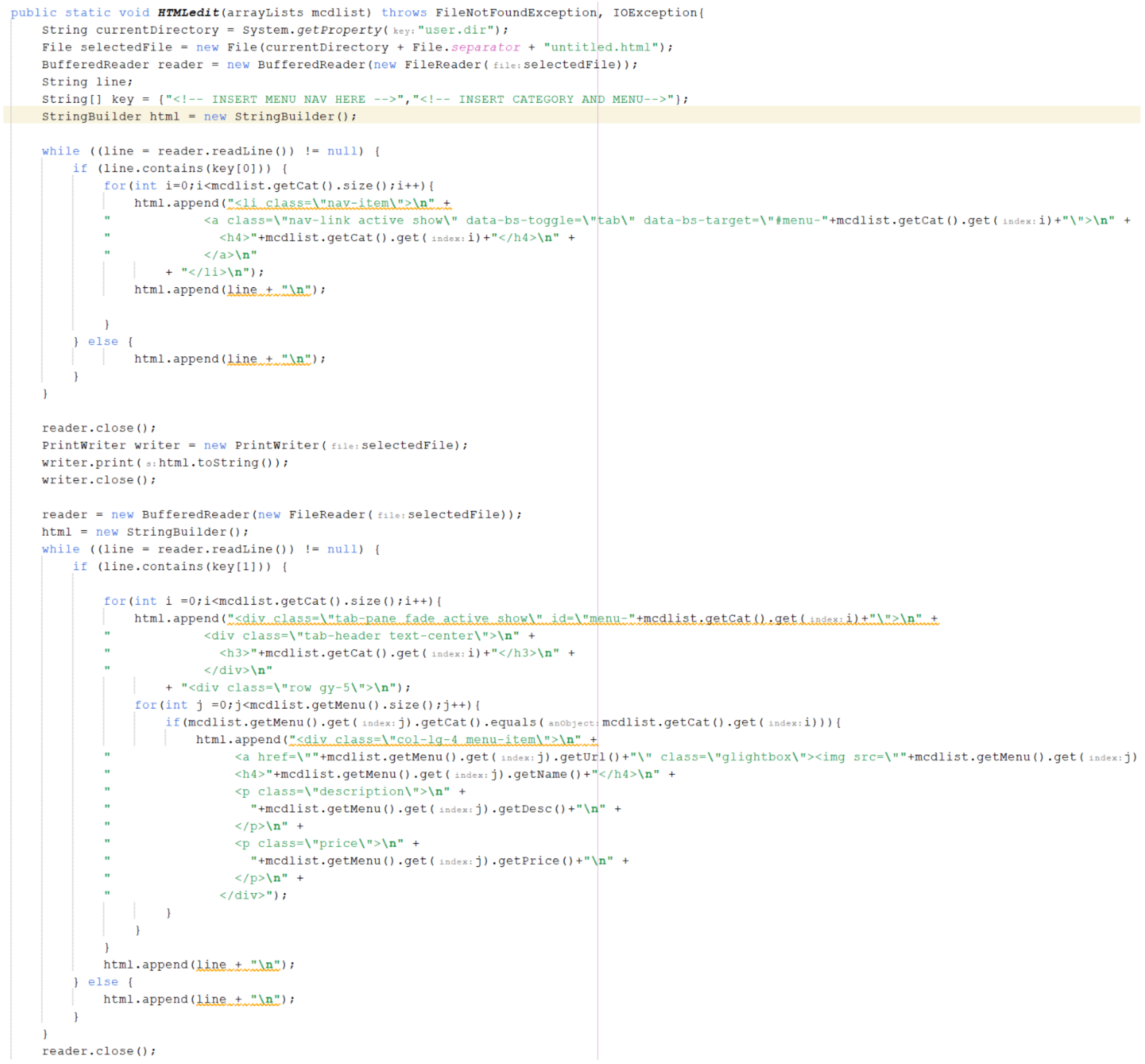




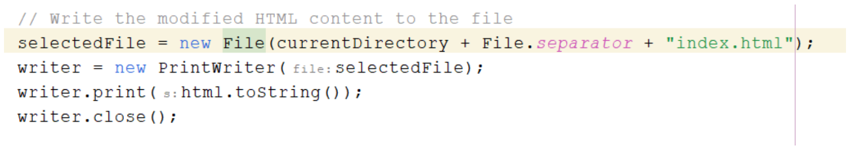
```
private arrayLists menuDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    int i = menuList.getSelectedIndex();
    // If an item is selected in the menu list
    if(i!=-1){
        for(int j=0;j<mcdlist.getCat().size();j++){
            // If the selected item is a category, remove it and all the menu items in that category from the arrayLists object
            if(display.get( index: i).equals( anObject: mcdlist.getCat().get( index: j))){
                catDropdown.removeItemAt( anIndex: j);
                for(int k=mcdlist.getMenu().size()-1;k>=0;k--){
                    // If the menu item is in the selected category, remove it from the arrayLists object
                    if(mcdlist.getMenu().get( index: k).getCat().equals( anObject: mcdlist.getCat().get( index: j))){
                        mcdlist.getMenu().remove( index: k);
                    }
                }
                mcdlist.getCat().remove( index: j);
                updateMenuList(mcdlist);
                return mcdlist;
            }
        }
        for(int j=mcdlist.getMenu().size()-1;j>=0;j--){
            // If the selected item is a menu item, remove it from the arrayLists object
            if(display.get( index: i).substring( beginIndex: 3).equals( anObject: mcdlist.getMenu().get( index: j).getName())){
                mcdlist.getMenu().remove( index: j);
            }
        }
        updateMenuList(mcdlist);
    }
    return mcdlist;
}
```

It checks if an item is selected, if true, determines if the item is a category or a menu item. If it's a category, it removes the category from the dropdown menu, removes all menu items in that category from the "arrayLists" object, and removes the category from the "arrayLists" object. If it's a menu item, it removes the menu item from the "arrayLists" object.

Creating the Webpage



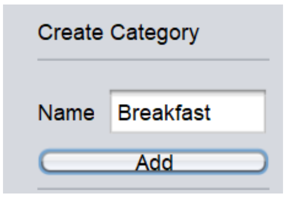
```
public static void HTMLedit(arrayLists mcdlist) throws FileNotFoundException, IOException{
    String currentDirectory = System.getProperty( key: "user.dir");
    File selectedFile = new File(currentDirectory + File.separator + "untitled.html");
    BufferedReader reader = new BufferedReader(new FileReader( file: selectedFile));
    String line;
    String[] key = {"<!-- INSERT MENU NAV HERE -->","<!-- INSERT CATEGORY AND MENU-->"};
    StringBuilder html = new StringBuilder();

    while ((line = reader.readLine()) != null) {
        if (line.contains(key[0])) {
            for(int i=0;i<mcdlist.getCat().size();i++){
                html.append("<li class=\"nav-item\">\n" +
            "            <a class=\"nav-link active show\" data-bs-toggle=\"tab\" data-bs-target=\"#menu-"+mcdlist.getCat().get( index: i)+"\">\n" +
            "              <h4>"+mcdlist.getCat().get( index: i)+"</h4>\n" +
            "            </a>\n"
                    + "</li>\n");
                html.append(line + "\n");

            }
        } else {
                html.append(line + "\n");
        }
    }

    reader.close();
    PrintWriter writer = new PrintWriter( file: selectedFile);
    writer.print( s: html.toString());
    writer.close();

    reader = new BufferedReader(new FileReader( file: selectedFile));
    html = new StringBuilder();
    while ((line = reader.readLine()) != null) {
        if (line.contains(key[1])) {

            for(int i =0;i<mcdlist.getCat().size();i++){
                html.append("<div class=\"tab-pane fade active show\" id=\"menu-"+mcdlist.getCat().get( index: i)+"\">\n" +
            "            <div class=\"tab-header text-center\">\n" +
            "              <h3>"+mcdlist.getCat().get( index: i)+"</h3>\n" +
            "            </div>\n"
                    + "<div class=\"row gy-5\">\n");
                for(int j =0;j<mcdlist.getMenu().size();j++){
                    if(mcdlist.getMenu().get( index: j).getCat().equals( anObject: mcdlist.getCat().get( index: i))){
                        html.append("<div class=\"col-lg-4 menu-item\">\n" +
            "                <a href=\""+mcdlist.getMenu().get( index: j).getUrl()+"\" class=\"glightbox\"><img src=\""+mcdlist.getMenu().get( index: j)
            "                <h4>"+mcdlist.getMenu().get( index: j).getName()+"</h4>\n" +
            "                <p class=\"description\">\n" +
            "                  "+mcdlist.getMenu().get( index: j).getDesc()+"\n" +
            "                </p>\n" +
            "                <p class=\"price\">\n" +
            "                  "+mcdlist.getMenu().get( index: j).getPrice()+"\n" +
            "                </p>\n" +
            "              </div>");
                    }
                }
            }
            html.append(line + "\n");
        } else {
            html.append(line + "\n");
        }
    }
    reader.close();
```

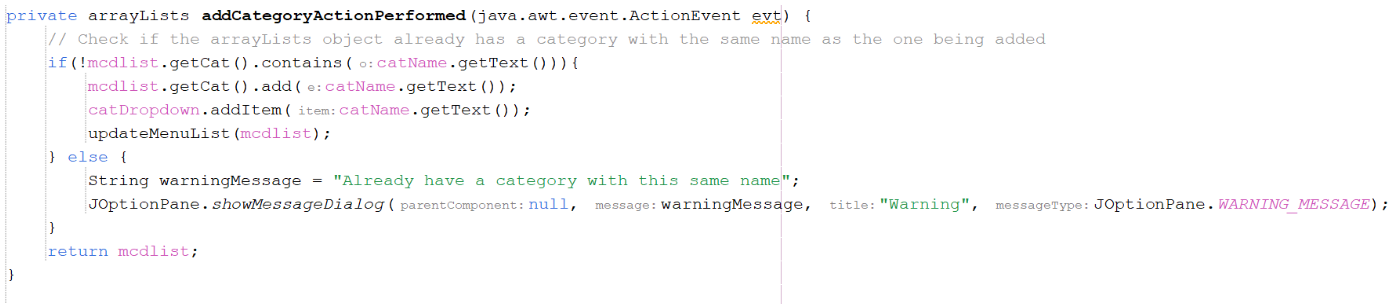


```
// Write the modified HTML content to the file
selectedFile = new File(currentDirectory + File.separator + "index.html");
writer = new PrintWriter( file: selectedFile);
writer.print( s: html.toString());
writer.close();
```

This code modifies an HTML file by replacing certain lines with new content. It reads the file line by line and checks if a line contains a specific string.It replaces the line with new conten. After all lines have been processed, the modified HTML content is written back to the file.

Adding new Categories / Menu items

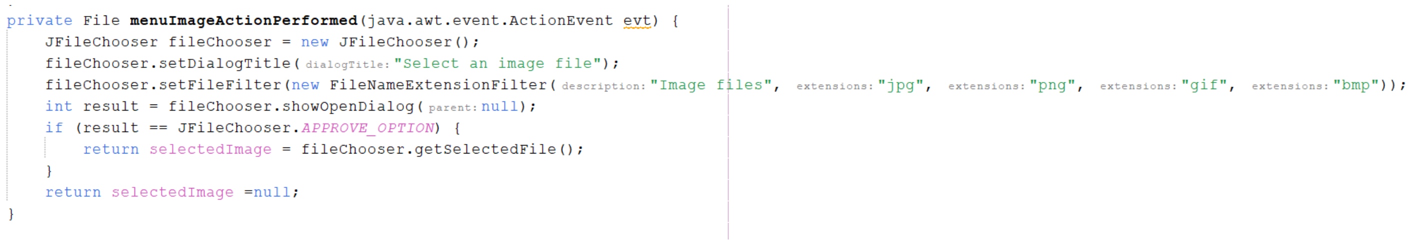




```
private arrayLists addCategoryActionPerformed(java.awt.event.ActionEvent evt) {
    // Check if the arrayLists object already has a category with the same name as the one being added
    if(!mcdlist.getCat().contains( o: catName.getText())){
        mcdlist.getCat().add( e: catName.getText());
        catDropdown.addItem( item: catName.getText());
        updateMenuList(mcdlist);
    } else {
        String warningMessage = "Already have a category with this same name";
        JOptionPane.showMessageDialog( parentComponent: null,  message: warningMessage,  title: "Warning",  messageType: JOptionPane.WARNING_MESSAGE);
    }
    return mcdlist;
}
```

This method checks if a category with the same name already exists in the arrayLists object. If it does not exist, the method adds the new category to the arrayLists object and the catDropdown object, and updates the menu list. If the category does exist, the method displays a warning message to the user.

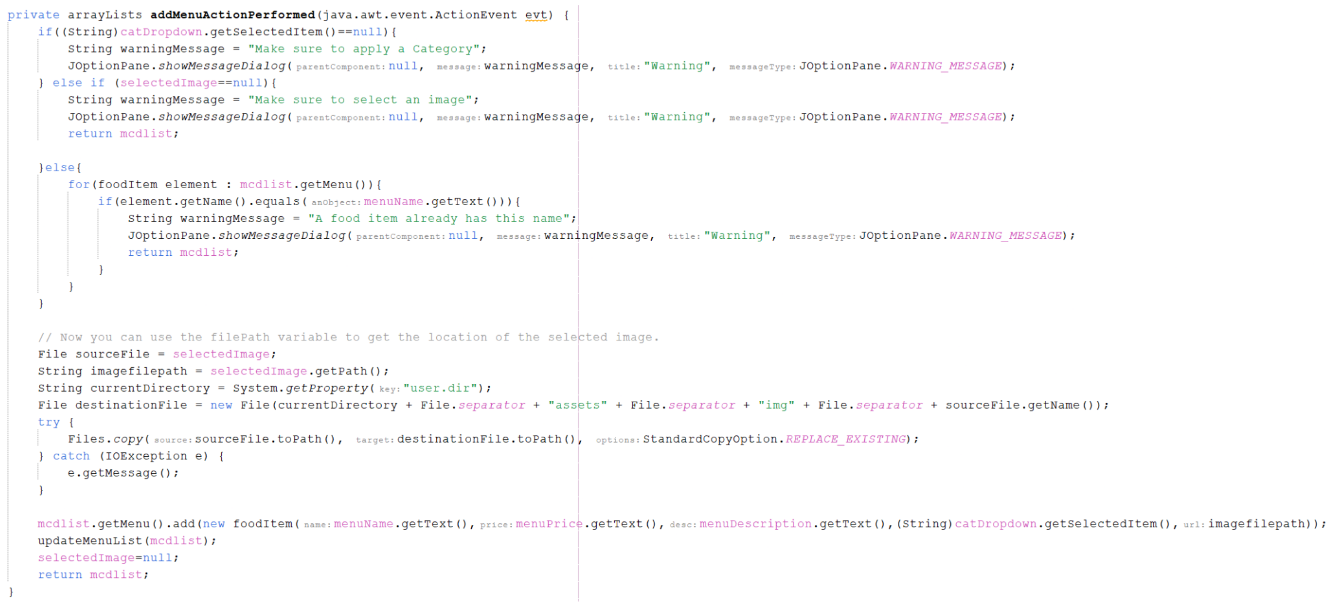
Selecting Image for Menu items



```
private File menuImageActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle( dialogTitle: "Select an image file");
    fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Image files",  extensions: "jpg",  extensions: "png",  extensions: "gif",  extensions: "bmp"));
    int result = fileChooser.showOpenDialog( parent: null);
    if (result == JFileChooser.APPROVE_OPTION) {
        return selectedImage = fileChooser.getSelectedFile();
    }
    return selectedImage =null;
}
```

Creates a file chooser that allows the user to select an image file, then returns it.

Adding Menu items



```java
private arrayLists addMenuActionPerformed(java.awt.event.ActionEvent evt) {
    if((String)catDropdown.getSelectedItem()==null){
        String warningMessage = "Make sure to apply a Category";
        JOptionPane.showMessageDialog( parentComponent: null,  message: warningMessage,  title: "Warning",  messageType: JOptionPane.WARNING_MESSAGE);
    } else if (selectedImage==null){
        String warningMessage = "Make sure to select an image";
        JOptionPane.showMessageDialog( parentComponent: null,  message: warningMessage,  title: "Warning",  messageType: JOptionPane.WARNING_MESSAGE);
        return mcdlist;

    }else{
        for(foodItem element : mcdlist.getMenu()){
            if(element.getName().equals( anObject: menuName.getText())){
                String warningMessage = "A food item already has this name";
                JOptionPane.showMessageDialog( parentComponent: null,  message: warningMessage,  title: "Warning",  messageType: JOptionPane.WARNING_MESSAGE);
                return mcdlist;
            }
        }
    }

    // Now you can use the filePath variable to get the location of the selected image.
    File sourceFile = selectedImage;
    String imagefilepath = selectedImage.getPath();
    String currentDirectory = System.getProperty( key: "user.dir");
    File destinationFile = new File(currentDirectory + File.separator + "assets" + File.separator + "img" + File.separator + sourceFile.getName());
    try {
        Files.copy( source: sourceFile.toPath(),  target: destinationFile.toPath(),  options: StandardCopyOption.REPLACE_EXISTING);
    } catch (IOException e) {
        e.getMessage();
    }

    mcdlist.getMenu().add(new foodItem( name: menuName.getText(), price: menuPrice.getText(), desc: menuDescription.getText(),(String)catDropdown.getSelectedItem(), url: imagefilepath));
    updateMenuList(mcdlist);
    selectedImage=null;
    return mcdlist;
}
```

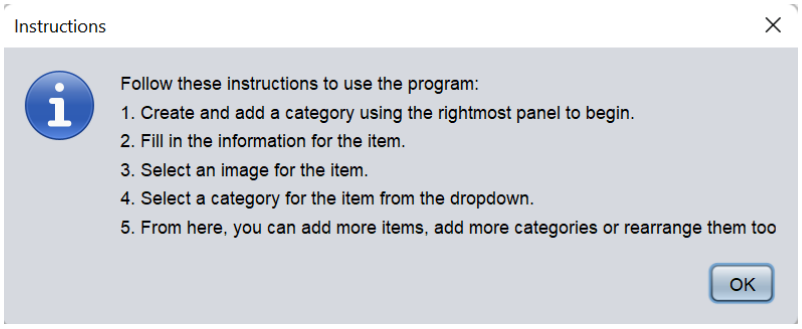
The function checks if the selected category is null or if an image has been selected. If either of these conditions is true, a warning message is displayed to the user. The function also checks if the name of the new food item already exists in the menu list. If it does, another warning message is displayed. Else, the function copies the selected image file to "assets/img" in the current working directory, and adds the new food item to the menu list.

Classes

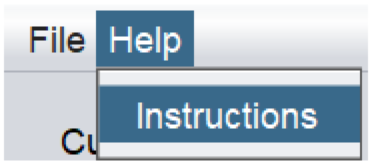
```java
package menugenerator;

import java.io.Serializable;

public class foodItem implements Serializable {
    public String name="";
    public String price="";
    public String desc="";
    public String cat="";
    public String url="";

    public foodItem(String name, String price, String desc, String cat, String url){
        this.name = name;
        this.price = price;
        this.desc = desc;
        this.cat = cat;
        this.url = url;
    }
    public String getName(){
        return name;
    }
    public String getPrice(){
        return price;
    }
    public String getDesc(){
        return desc;
    }
    public String getCat(){
        return cat;
    }
    public void setCat(String cat){
        this.cat=cat;
    }
    public String getUrl(){
        return url;
    }
}
```

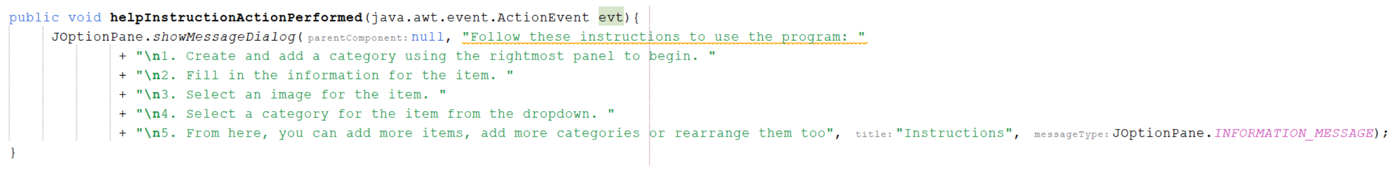
The foodItem class is a Java class that implements the Serializable interface and has instance variables (name, price, desc, cat, and url) of type string, as well as getter and setter methods.

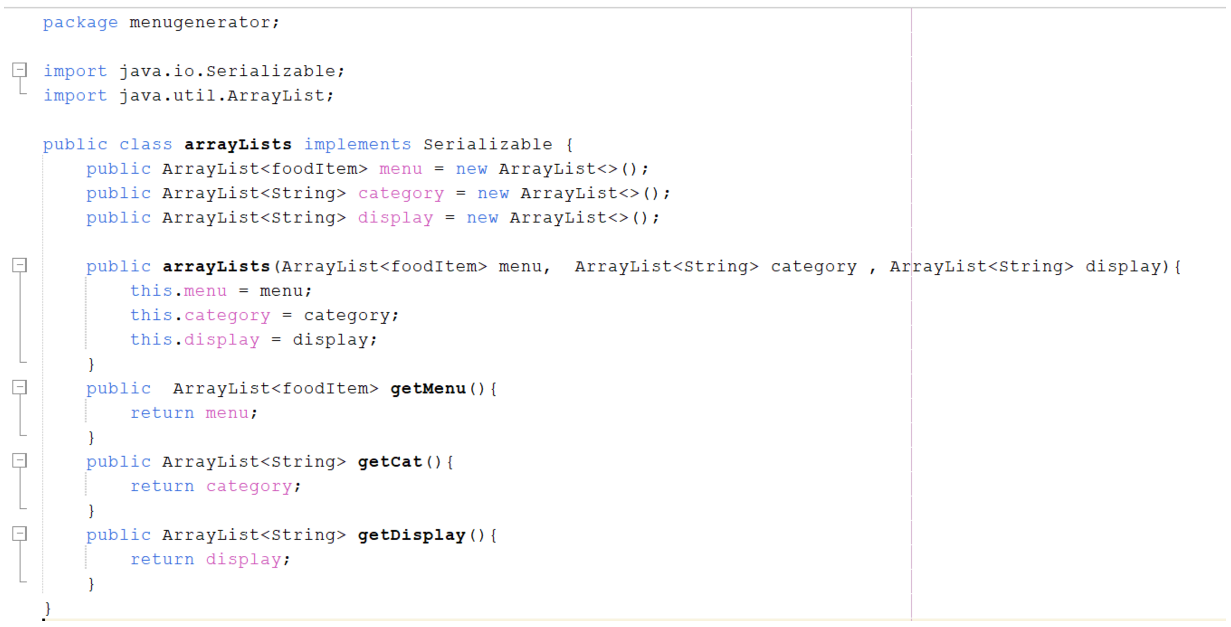
The class could be modified to store additional information about a food item, such as ingredients or nutritional information.


Instructions

Follow these instructions to use the program:
1. Create and add a category using the rightmost panel to begin.
2. Fill in the information for the item.
3. Select an image for the item.
4. Select a category for the item from the dropdown.
5. From here, you can add more items, add more categories or rearrange them too




```
public void helpInstructionActionPerformed(java.awt.event.ActionEvent evt){
    JOptionPane.showMessageDialog( parentComponent: null, "Follow these instructions to use the program: "
            + "\n1. Create and add a category using the rightmost panel to begin. "
            + "\n2. Fill in the information for the item. "
            + "\n3. Select an image for the item. "
            + "\n4. Select a category for the item from the dropdown. "
            + "\n5. From here, you can add more items, add more categories or rearrange them too", title: "Instructions", messageType: JOptionPane.INFORMATION_MESSAGE);
}
```
Dialog popup for instructions.



```
package menugenerator;

import java.io.Serializable;
import java.util.ArrayList;

public class arrayLists implements Serializable {
    public ArrayList<foodItem> menu = new ArrayList<>();
    public ArrayList<String> category = new ArrayList<>();
    public ArrayList<String> display = new ArrayList<>();

    public arrayLists(ArrayList<foodItem> menu,  ArrayList<String> category , ArrayList<String> display){
        this.menu = menu;
        this.category = category;
        this.display = display;
    }
    public  ArrayList<foodItem> getMenu(){
        return menu;
    }
    public ArrayList<String> getCat(){
        return category;
    }
    public ArrayList<String> getDisplay(){
        return display;
    }
}
```

ArrayLists implements the Serializable interface, allowing it to be written to a file. It has three instance variables:

- menu, an ArrayList of foodItem objects.
- category, an ArrayList of strings.
- display, an ArrayList of strings.

The" arrayLists" class has three getter methods.

Words: 999