Project #4 (140 points)

Due Date

Monday, November 21, by 11:59pm.

Submission

- Submit **a single zipped file** to Canvas. The zipped file MUST include the following grading items.
  (1) Source folder **src**, including all packages/folders listed below. [115 points]
    a) All Java file, i.e., *.java; these include all Java classes and controller classes. The file names of the controller classes MUST contain the key word "Controller", or you will **lose 2 points**.
    b) All fxml files; the file names of the fxml files MUST contain the keyword "View", or you will **lose 2 points.**
    c) All images used in this project.
  (2) A JUnit test class for BuildYourOwn class. [10 points]
  (3) Class diagram [10 points]
  (4) Javadoc [5 points]
- The submission button on Canvas will disappear after **November 21, 11:59pm**. It is your responsibility to ensure your Internet connection is good for the submission. **You get 0 points** if you do not have a submission on Canvas. Submitting the project through the email will not be accepted.

Project Description

RU Pizzeria sells pizzas. Your team will develop a software for the Pizzeria to manage the orders. Your team must use JavaFX to develop the GUIs for taking, placing, and cancelling orders. The Pizzeria offers 2 different styles of pizzas, Chicago style and New York style. The Pizzeria uses different pie crusts depending on the pizza flavors and styles. There are 3 specialty pizza flavors, Deluxe, BBQ Chicken, and Meatzza. All specialty pizzas are customized with specific crusts and toppings. Customers can order Build Your Own pizza and choose the toppings they like, with a maximum of 7 toppings. The store staff will be the one using the software to take the orders from the customers. Each order is uniquely identified by an order number generated by the system. For simplicity, the system will not keep track of the dates of the orders. The store manager has given you the list of requirements below.

(1) The Pizzeria offers Chicago style and New York style pizzas with 3 different sizes: small, medium, or large. Each pizza style or flavor uses a different pizza crust. The recipes for each of the pizza flavors are listed below.

Pizza Flavor	Chicago style Crust	New York Style Crust	Toppings	Price
Deluxe	Deep Dish	Brooklyn	Sausage, pepperoni, green pepper, onion, mushroom	Small: $14.99 Medium: $ 16.99 Large: $18.99
BBQ Chicken	Pan	Thin	BBQ chicken, green pepper, provolone, cheddar	Small: $13.99 Medium: $ 15.99 Large: $17.99
Meatzza	Stuffed	Hand tossed	Sausage, pepperoni, beef, ham	Small: $15.99 Medium: $ 17.99 Large: $19.99
Build your own	Pan	Hand tossed	• Customizable, up to 7 toppings • Add $1.59 for each additional topping	Small: $8.99 Medium: $ 10.99 Large: $12.99

(2) The system shall allow the store staff to choose the pizza style, flavors, and sizes, but not the crusts.
(3) Upon the selection of the pizza flavor, the system shall display the image of the selected pizza, the list of store-customized toppings, and a list of sizes to choose from.
(4) If "Build your own" pizza is chosen, the system shall display a list of toppings for customization. The store maintains at least 13 different toppings every day. The store staff shall be able to customize the pizza by adding or removing the toppings, with a maximum of 7 additional toppings.
(5) The system shall display a running subtotal with 2 decimal places on the ordering pizza page.
(6) The system shall allow the store staff to add multiple pizzas to the same order and remove selected pizzas from the order.
(7) The store staff shall be able to check the detail of the current order before placing the order. The system will keep track of the current order in a shopping cart. The order details in the shopping cart shall include the list of pizzas. Each pizza shall include the pizza style, crust, list of toppings, subtotal for each pizza, total amount of the pizzas in the order, sales tax amount and the order total, which is the total amount plus sales tax. The tax rate is 6.625%.
(8) The store staff shall be able to remove a pizza or remove all pizzas in the shopping cart. While the staff is removing pizzas from the shopping cart, the system shall update the total amount, sales tax, and the order total accordingly.
(9) The system shall be able to keep track of all the store orders, allow the store staff to browse the store orders and cancel an order. These shall include displaying all the store orders by the order numbers, the order total for each order with 2 decimal places, and the list of pizzas in each order.
(10) The system shall be able to export the store orders and save them in a text file, which includes a list of store orders. Each store order shall include the order number, the list of pizzas ordered, and the order total.

Project Requirements

1. You MUST follow the Coding Standard and Ground Rules posted on Canvas under Week #1 in the "Modules". **You will lose points** if you are not following the rules.
2. You are responsible for following the Academic Integrity Policy. See the **Additional Note #13** in the syllabus.
3. All methods should not exceed 40 lines, or **-2 points** for each violation, **maximum 5 points off**.
4. Each Java class must go in a separate file. **-2 points** if you put more than one Java class into a file.
5. You MUST set the titles of all the "stages" (titles for the windows) or **-2 points each.**
6. You are **NOT ALLOWED to use System.out** (write to console) or **System.in** (read from console) ANYWHERE in ALL CLASSES. All read and write must be done through the GUIs, **or you will lose 3 points for each violation, with a maximum of losing 10 points**.
7. File output performed for exporting store orders should be done in the StoreOrder class, or **-5 points.**
8. Your software must provide at least 5 Views listed below, **-5 points** for each View missing. The user can navigate between the Views to add/remove pizzas to/from an order, review/clear/place the order, check all store orders, and cancel a store order.
   - **MainView**, which shall provide the options of choosing the pizza styles, checking the current order in the shopping cart, and managing the store orders.
   - **Chicago Style Pizza Ordering View**, which displays the image of the chosen pizza flavor, sizes to choose from, the crust of the pizza, a list of preset toppings, a list of additional toppings, the running total of the pizza while the user is customizing the toppings. You MUST use the same View for all pizza flavors. If you create a View for each pizza flavor, you will **lose 10 points.**
   - **New York Style Pizza Ordering View**, which displays the image of the chosen pizza flavor, sizes to choose from, the crust of the pizza, a list of preset toppings, a list of additional toppings, the running total of the pizza while the user is customizing the toppings. You MUST use the same View for all pizza flavors. If you create a View for each pizza flavor, you will **lose 10 points.**

- **Current order View**. This is the shopping cart that displays the order number, the list of pizzas with the pizza style, crust, the list of toppings, subtotal, sales tax, and order total. All dollar amounts must be displayed with 2 decimal places. The user can review the order, remove the selected pizza, clear the order, or place the order.
- **Store orders View**, which displays all the orders placed so far. The View shall list the details of each order and display the total amount of each order, with 2 decimal places. The user can select an order and cancel the order. The View shall display the remaining orders and the total amount correctly after the cancellation. The user can also export the store orders to an external text file, which shall include the details of every order, consistent with the details displayed on the View.

9. For each View, you must create a .fxml file and its associated controller class. That is, you will have 5 .fxml files and 5 controller.java files. **-5 points** if this is not done properly.
10. You can use any JavaFX UI controls or containers to design your Views. However, you MUST use ComboBox, ImageView and ListView, or **-5 points** for each violation.
11. You can use any Java library classes. However, you MUST meet the following project requirements.
    - All the instance variables defined in the controller classes must be "private", **-2 points** for each violation.
    - Must include the **abstract Pizza class** below. You cannot add/change the instance variables below, or **-2 points.** You can add "static final" data items (constants.) You can add additional methods if necessary. All flavors of pizzas should be a subtype of Pizza class; for example, `public class Deluxe extends Pizza`. You should have 4 subclasses extending the Pizza class, including Deluxe, BBQChicken, Meatzza and BuildYourOwn. **-5 points** for each subclass not implemented or not used. Must implement the Customizable interface below for adding/removing toppings.

```
public abstract class Pizza implements Customizable {
    private ArrayList<Topping> toppings;
    private Crust crust;
    private Size size;
    public abstract double price();
}

public interface Customizable {
   boolean add(Object obj);
   boolean remove(Object obj);
}
```

    - Must include the PizzaFactory interface below. Must include the ChicagoPizza class and NYPizza class below, and they must implement the PizzaFactory interface, **-5 points** for each class missing. Their responsibilities are creating (making) the pizzas with the crusts and toppings. They should be used in the controller classes to create an instance of Pizza class based on the chosen flavor. **That is, these are the ONLY classes that can "new" an instance of the subclasses of the Pizza class**. This is to hide the details of creating the pizza objects and allow flexibility to add new pizza flavors or new pizza styles in the future, this is a **design pattern** called **"Abstract Factory".** For example, the two statements below create a Chicago style Deluxe pizza.

```
PizzaFactory pizzaFactory = new ChicagoPizza();
Pizza pizza = pizzaFactory.createDeluxe();

public interface PizzaFactory {
   Pizza createDeluxe();
   Pizza createMeatzza();
   Pizza createBBQChicken();
   Pizza createBuildYourOwn();
}

public class ChicagoPizza implements PizzaFactory{ }

public class NYPizza implements PizzaFactory { }
```

You CANNOT add any instance variables, static final constants, or other methods to ChicagoPizza or NYPizza class, or you will **lose 5 points.** You CANNOT use the subclasses such as Deluxe, BBQChicken, Meatzza and BuildYourOwn classes, as data types in ALL other Java classes except the ChicagoPizza class and NYPizza class. **-5 points** for each violation. You should ONLY use the Pizza class as the general data type in all other Java classes for any instance of pizzas.

- Must include an **Order class**. An instance of this class has a unique order number and keeps the list of instances of Pizza class. Whenever an order object is created, the system generates a serial number for the order, and the number is a unique integer; **-5 points** if the class is not implemented or not used. All instance variables must be private or **lose 2 points**. Must implement the Customizable interface to add/remove pizzas.

```
public class Order implements Customizable { }
```

- Must include an **StoreOrders class**. An instance of this class keeps the list of orders placed by the user. This class keep track of the order numbers; **-5 points** if the class is not implemented or not used. All instance variables must be private or **lose 2 points**. You must include an **export()** method to save the store orders to an external text file, **-5 points** if this is not done in this class. Must implement the Customizable interface to add/remove orders.

```
public class StoreOrder implements Customizable { }
```

12. Create a JUnit test class for BuildYourOwn class, which should be a subclass of the Pizza class, and test the price() method. Use the Black-Box testing technique to design the test cases and ensure the price() method performs properly with different numbers of toppings and different sizes. The JUnit test class and test cases are **worth 10 points**. You MUST comment your test cases, but DO NOT need to generate Javadoc for the JUnit test class.
13. You are required to **generate the Javadoc** after you properly commented your code. Your Javadoc must include the documentations for the constructors, private methods and public methods of all Java classes (*.java files.) You **must comment ALL Java classes, including the Main.java and ALL controller classes.** They must be included in the Javadoc. DO NOT include the *.fxml files, which are NOT java files. DO NOT include the JUnit test class. Generate the Javadoc in a single folder and include it in your project folder to be submitted to Canvas. You are responsible to **double check** your Javadoc after you generated them. The grader will navigate the Javadoc with the "index.html". You will **lose 5 points** for not including the Javadoc, OR, the grader cannot navigate your Javadoc through the "index.html".
14. Create a Class diagram for this project to document your design. You must include all Java classes you created in this project. DO NOT include the library classes from Java or JavaFX. DO NOT include the JUnit test class and .fxml files. Each rectangle includes the class name and the instance variables only; no need to include the constructors and methods. The class diagram **is worth 10 points**.

System Testing

1. You are responsible to thoroughly test your software and ensure your software is meeting the requirements listed above. You will **lose 2 points** for each incorrect implementation, incorrect amount or incorrect information shown on the GUIs.
2. Your software must always run in a sane state and **should not crash in any circumstances**. You must catch all Java Exceptions. Your program shall continue to run until the user stops the program execution or closes the window. **You will lose 2 points** for each exception not caught.