Project #4 (170 points)

Due Date

Monday, November 18, by 11:59 pm.

Submission

(1) Zip the project folder and submit the zipped file to Canvas.

(2) The zip file must include the following grading items.

- Source folder **src**, including all packages/folders listed below. [**140 points**]
  a) All Java files, *.java; the names of the Java files for the controllers MUST contain the key word "controller", or you will **lose 2 points**.
  b) All FXML files; the file names MUST contain the keyword "view," or you will **lose 2 points.**
  c) All image files and FXML files must be stored under the "resources" folder.
- A JUnit test class for BuildYourOwn class. [**15 points**]
- Class diagram [**10 points**]
- Javadoc [**5 points**]

(3) The submission button on Canvas will disappear after **November 18, 11:59 pm**. Do not wait until the last minute to submit the project. You are responsible for ensuring the project is well-received in Canvas by the due date and time. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through emails or other methods will not be accepted.**

Project Description

Your team will develop a software system for RU Pizzeria to manage the orders. Your team must use JavaFX to develop the GUIs for taking, placing, and canceling orders. The pizzeria offers two styles of pizzas: Chicago and New York. The pizzeria uses different pie crusts depending on the pizza type and style. Three specialty pizzas are Deluxe, BBQ Chicken, and Meatzza. All specialty pizzas are customized with specific crusts and toppings. Customers cannot customize the specialty pizzas. However, customers can order Build Your Own pizzas and customize the toppings, up to seven toppings. The store staff will use the software to take 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 and customer information 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 in three different sizes: small, medium, or large. The pizza crust is different depending on the style and type of the pizza. The recipes for each of the pizza types are listed below.

Pizza Type	Chicago style Crust	New York Style Crust	Toppings	Price
Deluxe	Deep Dish	Brooklyn	Sausage, pepperoni, green pepper, onion, mushroom	Small: $16.99 Medium: $ 18.99 Large: $20.99
BBQ Chicken	Pan	Thin	BBQ chicken, green pepper, provolone, cheddar	Small: $14.99 Medium: $ 16.99 Large: $19.99
Meatzza	Stuffed	Hand-tossed	Sausage, pepperoni, beef, ham	Small: $17.99 Medium: $ 19.99 Large: $21.99
Build your own	Pan	Hand-tossed	• Customizable, up to 7 toppings • Add $1.69 for each topping	Small: $8.99 Medium: $ 10.99 Large: $12.99

(2) The system shall allow the staff to choose the pizza style, type, and size, not the crust.
(3) Upon selecting the pizza style and type, the system shall display the image of the pizza selected, the list of toppings come with the pizza, and the sizes to choose from.
(4) If "Build your own" pizza is chosen, the system shall display a list of available toppings for customization. The store maintains a minimum of thirteen toppings every day. The staff shall be able to customize the pizza by adding or removing the toppings, with a maximum of seven toppings.
(5) The system shall display a running subtotal dynamically with two decimal places on the ordering pizza page while the staff is adding or removing the toppings.
(6) The system shall allow the staff to add multiple pizzas to the same order and remove selected pizzas from the order before the staff places the order.
(7) The staff shall be able to check the details of the current order before placing the order.
(8) The order details shall include the list of pizzas added, the total amount of the pizzas, the sales tax, and the order total, which is the total amount plus sales tax. The tax rate in New Jersey is 6.625%. Each pizza in the order shall consist of the pizza style, crust, list of toppings, and the subtotal amount of the pizza.
(9) The staff shall be able to remove a pizza or all pizzas from the current order. While the staff removes pizzas from the current order, the system shall update the total amount, sales tax, and the order total on the screen immediately after the removal.
(10) The system shall be able to keep track of all the orders placed, including browsing the orders and canceling an order. The system shall display the orders by the order numbers, the order total for each order with two decimal places, and the list of pizzas in each order.
(11) The system shall be able to export the orders placed and save them in a text file; each order shall include the order number, the list of pizzas ordered, and the order total.

Project Requirements

1. You MUST follow the Coding Standard posted on Canvas under Modules/Week #1. **You will lose points** if you violate the rules listed in the coding standard.
2. You are required to follow the Academic Integrity Policy. See the **Additional Note #13** in the syllabus posted on Canvas. If your team uses a repository hosted on a public website, you MUST set the repository to private. Setting it to public is considered as violating the academic integrity policy. The consequences of violation of the Academic Integrity Policy are: **(i) all parties involved receive 0 (zero) on the project, (ii) the violation is reported, and (iii) a record on your file of this violation.**
3. You MUST set the titles of all the "stages" (windows) or **-2 points each.**
4. You are **NOT ALLOWED** to use the **System.out** (write to console) or **System.in** (read from console) in any Java class. All read and write must be done through the GUIs, **or you will lose 3 points for each violation,** with a **maximum of 10 points**.
5. **Navigation design**. Your software must provide at least THREE different "views" for ordering pizzas, **-5 points** for each violation. The user shall be able to view/add/remove multiple pizzas of a current order. -**10 points** if this is not done properly. The users shall be able to view the details of all the orders placed and cancel an order, -**10 points** if this is not done properly.
6. The GUI requirements are listed below. **-3 points** for each requirement not met.
   - Provide the options of choosing the pizza styles, browsing the current order and managing the orders placed.
   - Provide the options for ordering Chicago-style or New York-style pizzas and display the image of the chosen pizza type, size, crust, the list of toppings and additional toppings, and the running total of the pizza while the user is customizing the toppings.

- Navigate the current order before placing it, including 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.
- Display the details of the orders placed. It 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.
- Export the orders to a text file, which shall include the details of every order consistent with the details displayed on the View**. -10 points** if the software cannot export the orders.

7. For each View, you must create a .fxml file and its associated Controller class. For example, if you have three .fxml files, you should have three Controller.java files. **-5 points** if this is not done properly.
8. You can use any JavaFX UI objects to design your Views. However, you MUST include Image Buttons, ComboBox and ListView, or **-5 points** for each violation.
9. You can use any Java library class. 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 each violation.** You can reuse the `List<E>` class you created in Project 2 as the data type instead of the `ArrayList<E>` for the toppings. You can add "static final" data items (constants.) and additional methods if necessary. All types of pizzas should be a subtype of Pizza class. You should define four subclasses extending the Pizza class: Deluxe, BBQChicken, Meatzza and BuildYourOwn. **-5 points** for each subclass not implemented or not used.

```
public abstract class Pizza {
    private ArrayList<Topping> toppings; //Topping is a Enum class
    private Crust crust;  //Crust is a Enum class
    private Size  size;   //Size is a Enum class
    public abstract double price();
}
```

   - Must include the `PizzaFactory interface` below; the `ChicagoPizza` class and `NYPizza` class must implement the `PizzaFactory` interface, **-5 points** for each class missing. Their responsibilities are creating (making) the pizza objects with the crusts and toppings. They should be used in the controller classes to create an instance of the Pizza class based on the chosen type. Only the instances of `PizzaFactory` **can "new" an object of the subclasses** of the Pizza class **or -5 points**. This design pattern, "**Abstract Factory**, " hides the details of creating the pizza objects, allowing flexibility to add new types or styles later. For example, the two statements below create a Chicago-style Deluxe pizza.

```
PizzaFactory pizzaFactory = new ChicagoPizza();
Pizza pizza = pizzaFactory.createDeluxe(); //create a Chicago-style deluxe pizza

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 `Deluxe`, `BBQChicken`, `Meatzza` and `BuildYourOwn` 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, or **-5 points.**

- Must include an **Order class**. An instance of this class has a unique order number and keeps the list of instances of the 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. You cannot add or change the instance variables, or **-2 points** for each violation.

```
public class Order {
    private int number; //order number
    private ArrayList<Pizza> pizzas; //can use List<E> instead of ArrayList<E>
}
```

10. 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 sizes. You must implement at least **5 test cases**. The JUnit test class and test cases are **worth 15 points**.
11. You must generate the Javadoc after you properly comment your code. Your Javadoc must include the documentation for the constructors and private and public methods of all Java classes (*.java files.) You **must comment on ALL Java classes and the Controller classes.** 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 for double-checking your Javadoc after you generate them.
12. Create a **Class diagram** for this project to document your design. You must include all Java classes you created for this project. DO NOT include the JavaFX/Java library classes and the JUnit test class. 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**.

Functional Testing

1. You are responsible for thoroughly testing your software and ensuring your software meets 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.