SEN2212
Data Structure and Algorithms II

Project Report

Group No: 5
Project Title: Restaurant Menu

Lab Section No: 901
Student ID: Arda Özdemir - Emir Sarı
Student Full Name: 2101584 - 2103185

1. Introduction

1.1 Purpose/Project Proposal

This project aims to produce a console application that serves as an interactable restaurant menu. The menu consists of 58 distinct items: 33 foods and 25 drinks. The user interacts with the application through the console.

- The user (customer) is provided with 4 options: display menu, search item, place order, and exit. The user is repeatedly prompted to choose 1 out of 4 options unless they decide to "exit" the program. Each option was carefully implemented to simulate a real-life restaurant experience.

    1. The "***Display menu***" option displays foods and drinks menu items in full detail with names, ingredients, kinds, content warnings, and cost information, sorted by kind.
    2. The "***Search item***" option lets the user search for an item by name. If the item is found, its details are returned; if not, the user is notified.
    3. The "***Place order***" option lets the user add items to their order while displaying their details. Finally, the user is informed of the total cost of their order.
    4. The "***Exit***" option terminates the program, meaning the user exits the restaurant.

- The console application consists of 15 classes:
    - ➔ 3 Enum Classes: DrinkKind, FoodKind, ContentWarnings
    - ➔ 11 General Purpose Concrete Classes: Food, FoodTree, FoodTreeNode, Drink, DrinkTree, DrinkTreeNode, IngredientsNode, IngredientsLinkedList, ContentWarningsNode, ContentWarningsLinkedList, Menu, Main.

  Common attributes such as name, ingredients, kind, content warnings, and cost; as well as common methods belonging to foods and drinks are defined in the "Food" class. "Drink" extends "Food" to inherit these common features, except kind. Food and drink kinds are defined separately in the "FoodKind" and "DrinkKind" enum classes since both categories have distinct kinds. ContentWarnings enum class is used to store content warnings related to menu items.

  Foods and drinks are stored as tree nodes, for this purpose, 2 different binary search trees and classes related to their implementation are created. Additionally, linked lists are used to store each item's content warnings and ingredients. This is due to the dynamic size property of linked lists. This way attributes can easily be managed for each item individually.

  Finally, the restaurant menu is populated in the Main class where each item is added in full detail. Firstly food and drink trees are constructed, and then both trees are added to the menu. User interactions and necessary calculations are also implemented in "Main". In the menu, ANSI color codes are used on the console screen to make it look aesthetically pleasing.

- Regarding the main skeleton of the project, here are a few examples of dynamic architecture:

    - ➔ multipleInsert(Food[] foods) and multipleInsert(Drink[] drinks) methods are defined to add all foods and drinks to trees at once. Creating an array of foods/drinks before insertion saved both time and space during coding.

    - ➔ A recursive search(Food food, FoodTreeNode t) function, in FoodTree class, gets a specific instance of Food (or Drink if the function is implemented in DrinkTree

class) and returns the result after a few iterations of comparisons between “food” and leaves of the tree.

➔ printMenu(FoodTree foodTree, DrinkTree drinkTree) function, in Menu class, gets both trees and separates the items by their FoodKind or DrinkKind to print the menu to the console. This separation is completed by traversing the trees and storing their items in arrays of “ArrayList”s. The fixed size property is appropriate for this use case as the numbers of constants in each enum are predefined.

1.2 Software Language/ Project Environment

During the development of this project, Java 16 and Eclipse IDE 4.26.0 were used.

1.3 Data Structures

The ***“Linked List”*** data structure was used to implement classes “ContentWarningLinkedList” and “IngredientsLinkedList”. Since each food/drink item might have different numbers of content warnings and ingredients, the implementation of arrays was not an option because of their fixed size property. Thus, we made use of the dynamic size of linked lists to store some attributes of each item.

The "***Binary Search Tree***" data structure was crucial in the integration of the storage of foods and drinks. “FoodTree” and “DrinkTree” classes were implemented using this data structure. It allowed for quicker search and order placement functionalities. The AVL Tree was a different option that could perform better in search efficiency. However, considering the scale of this project, the required allowance of time was much higher than the implementation of BST and the benefits would be minimal.

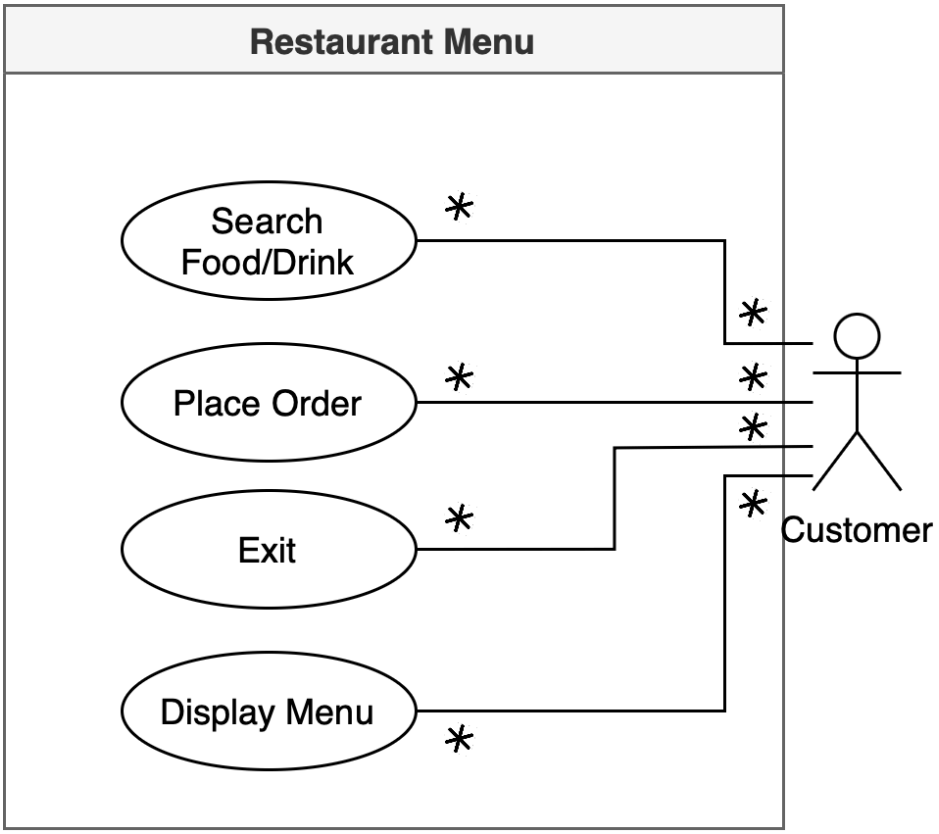
1.4 Work Partitioning

Name	*Role*	*Date*	*Description*
Emir Sarı 2103185	Back-end developer	21.05.2023 - 29.05.2023	I have done general research on method definitions and implementations. I coded concepts related to “foods”.
Arda Özdemir 2101584	Back-end developer	21.05.2023 - 29.05.2023	I found compatible functionalities and generic Java classes to implement. I coded drink-related concepts.

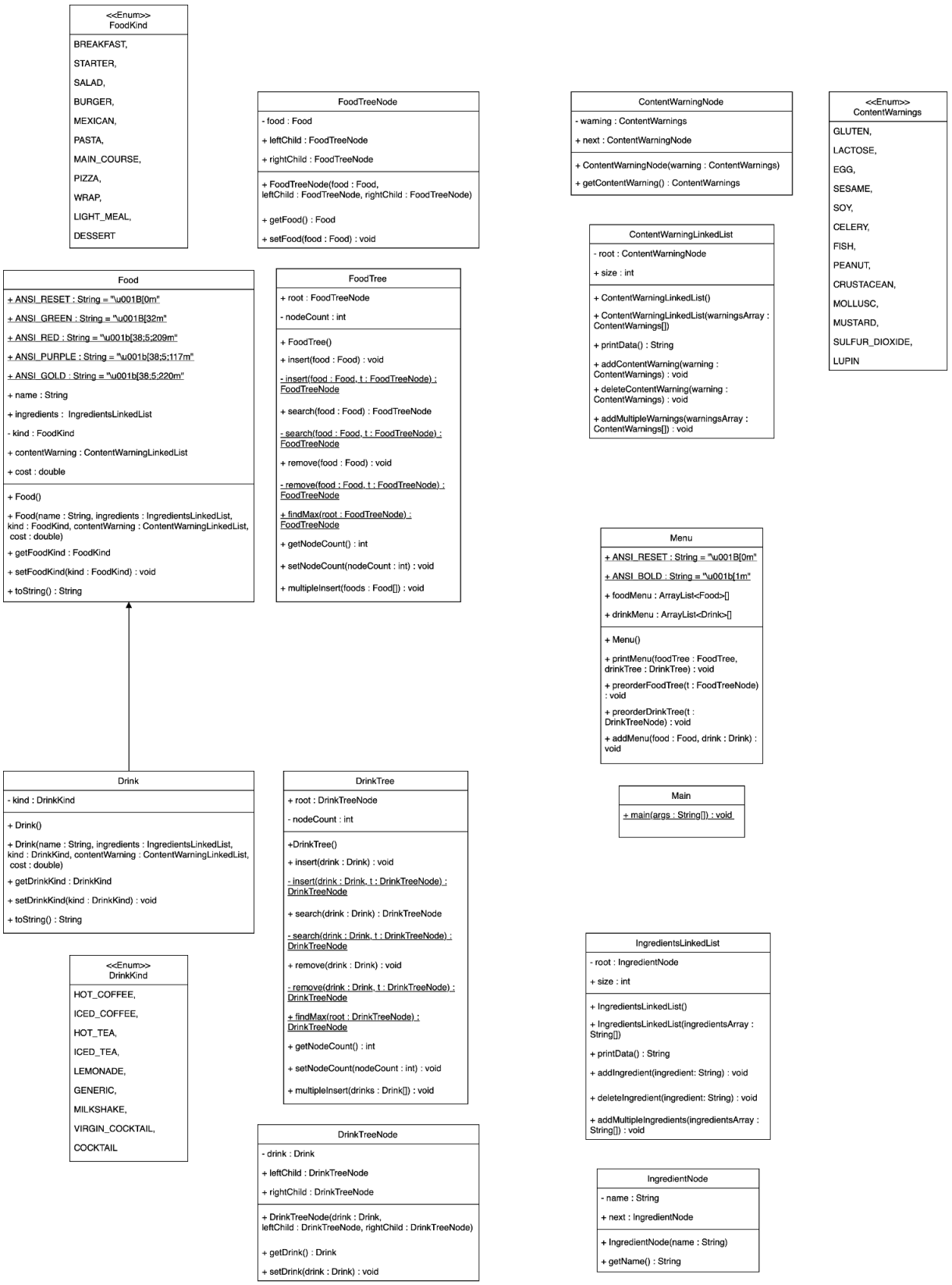
During the decision-making processes and UI design of the console application, the developers contributed to the project together. As the steps are cumulative, the tasks are done in sequence because classes and functions are required to be compatible with each other. For the process of coding the application, Github is used to cooperate. For the shared part of classes, there was high involvement of cooperation so that both developers were responsible for all of the data structures in the project.

2. Architectural Representation

2.1 Use Case Diagram



2.2 UML Class Diagram



3. Application

```java
public class ContentWarningLinkedList {

    private ContentWarningNode root;
    public int size = 0;

    // default constructor
    public ContentWarningLinkedList() {
        root = null;
        size = 0;
    }

    public ContentWarningLinkedList(ContentWarnings[] warningsArray) {
        root = null;
        size = 0;
        addMultipleWarnings(warningsArray);
    }

    public String printData() {

        ContentWarningNode current = root;
        String warnings = "";

        while (current != null) {

            if (current == root) {
                warnings += current.getContentWarning();
                // System.out.print(current.getContentWarning());
            } else {
                warnings += (", " + current.getContentWarning());
                // System.out.print(", " + current.getContentWarning());
            }
            current = current.next;

        }
        return warnings;
    }
    public void addContentWarning(ContentWarnings warning) {

        if (root == null) {
            // list is empty
            root = new ContentWarningNode(warning);
        } else {
            // list is not empty
            ContentWarningNode current = root;

            while (current.next != null) {
                current = current.next;
            }

            current.next = new ContentWarningNode(warning);
        }
        size++;
    }

    public void deleteContentWarning(ContentWarnings warning) {

        ContentWarningNode curr = root, prev = null;

        if (curr != null && curr.getContentWarning() == warning) {
            root = curr.next;
            size--;
            return;
        }

        while (curr != null && curr.getContentWarning() != warning) {
            prev = curr;
            curr = curr.next;
        }

        if (curr == null)
            return;

        prev.next = curr.next;
        size--;
    }

    public void addMultipleWarnings(ContentWarnings[] warningsArray) {

        for (int i = 0; i < warningsArray.length; i++) {
            addContentWarning(warningsArray[i]);
        }
    }
}
```

```java
public class ContentWarningNode {

    private ContentWarnings warning;
    public ContentWarningNode next;

    // default constructor
    public ContentWarningNode(ContentWarnings warning) {
        this.warning = warning;
        next = null;
    }

    // get content warning name
    public ContentWarnings getContentWarning() {
        return warning;
    }
}
```

```java
public enum ContentWarnings {

    GLUTEN,
    LACTOSE,
    EGG,
    SESAME,
    SOY,
    CELERY,
    FISH,
    PEANUT,
    CRUSTACEAN,
    MOLLUSC,
    MUSTARD,
    SULFUR_DIOXIDE,
    LUPIN;
}
```

Above are classes related to content warnings. An enum class is used to store different types of warnings. A traditional linked list class with its node class is created, and methods related to adding, removing, and displaying content warning information are implemented.

```java
public class IngredientsLinkedList {

    private IngredientNode root;
    public int size;

    // default constructor
    public IngredientsLinkedList() {
        root = null;
        size = 0;
    }

    public IngredientsLinkedList(String[] ingredientsArray) {
        root = null;
        size = 0;
        addMultipleIngredients(ingredientsArray);
    }

    public String printData(){

        IngredientNode current = root;
        String ingredients = "";

        while(current != null){

            if(current == root) {
                ingredients += current.getName();
                //System.out.print(current.getName());
            }
            else{
                ingredients += (", " + current.getName());
                //System.out.print(", " + current.getName());
            }
            current = current.next;


        }
        return ingredients;
    }
    public void addIngredient(String ingredient) {

        if(root == null) {
            // list is empty
            root = new IngredientNode(ingredient);
        }else {
            // list is not empty
            IngredientNode current = root;

            while(current.next != null) {
                current = current.next;
            }

            current.next = new IngredientNode(ingredient);
        }
        size++;
    }

    public void addMultipleIngredients(String[] ingredientsArray) {

        for(int i = 0; i < ingredientsArray.length; i++) {

            addIngredient(ingredientsArray[i]);
        }

    }
}
```

```java
public class IngredientNode {

    private String name;
    public IngredientNode next;

    // default constructor
    public IngredientNode(String name) {
        this.name = name;
        next = null;
    }

    // get ingredient name
    public String getName() {
        return name;
    }

}
```

```java
public void deleteIngredient(String ingredient)
{

    IngredientNode curr = root, prev = null;


    if (curr != null && curr.getName().equals(ingredient)) {
        root = curr.next;
        size--;
        return;
    }


    while (curr != null && !(curr.getName().equals(ingredient))) {
        prev = curr;
        curr = curr.next;
    }


    if (curr == null)
        return;

    prev.next = curr.next;
    size--;
}
```

Above are classes related to ingredients. A traditional linked list class with its node class is created, and methods related to adding, removing, and displaying ingredient information are implemented. Also, an additional method is provided to add multiple ingredients at once.

```java
public class Food{

    // text coloring
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_GREEN = "\u001B[32m";
    public static final String ANSI_RED = "\u001b[38;5;209m";
    public static final String ANSI_PURPLE = "\u001b[38;5;117m";
    public static final String ANSI_GOLD = "\u001b[38;5;220m";

    public String name;
    public IngredientsLinkedList ingredients;
    private FoodKind kind;
    public ContentWarningLinkedList contentWarning;
    public double cost;

    // default constructor
    public Food() {
        name = null;
        ingredients = null;
        kind = null;
        contentWarning = null;
        cost = 0.0;
    }


    public Food(String name, IngredientsLinkedList ingredients, FoodKind kind,
            ContentWarningLinkedList contentWarning, double cost){

        this.name = name;
        this.ingredients = ingredients;
        this.kind = kind;
        this.contentWarning = contentWarning;
        this.cost = cost;
    }

    public FoodKind getFoodKind() {
        return kind;
    }

    public void setFoodKind(FoodKind kind) {
        this.kind = kind;
    }

@Override
public String toString() {
    return ANSI_GREEN + name + ANSI_RESET + " " + ANSI_GOLD + "$" + cost + ANSI_RESET + "\n\t\t\t" + ANSI_PURPLE + "Ingredients: " +
           ANSI_RESET + ingredients.printData() + (((contentWarning != null) && (contentWarning.size > 0) ? "\n\t\t\t"+ ANSI_RED +
           "Content Warnings: "+ ANSI_RESET + contentWarning.printData():")"));
```

```java
public enum FoodKind {

    BREAKFAST,
    STARTER,
    SALAD,
    BURGER,
    MEXICAN,
    PASTA,
    MAIN_COURSE,
    PIZZA,
    WRAP,
    LIGHT_MEAL,
    DESSERT;

}
```

```java
public class Drink extends Food {

    private DrinkKind kind;

    public Drink() {
        name = null;
        ingredients = null;
        kind = null;
        contentWarning = null;
        cost = 0.0;
    }

    public Drink(String name, IngredientsLinkedList ingredients, DrinkKind kind,
            ContentWarningLinkedList contentWarning, double cost) {
        this.name = name;
        this.ingredients = ingredients;
        this.kind = kind;
        this.contentWarning = contentWarning;
        this.cost = cost;
    }


    public DrinkKind getDrinkKind(){
        return kind;
    }

    public void setDrinkKind(DrinkKind kind) {
        this.kind = kind;
    }

    @Override
    public String toString() {
        return ANSI_GREEN + name + ANSI_RESET + " " + ANSI_GOLD + "$" + cost + ANSI_RESET + "\n\t\t\t" + ANSI_PURPLE + "Ingredients: " +
    ANSI_RESET + ingredients.printData() + (((contentWarning != null) && (contentWarning.size > 0) ? "\n\t\t\t"+ ANSI_RED + "Content Warnings: "+ ANSI_RESET
    contentWarning.printData():")"));
```

```java
public enum DrinkKind {

    HOT_COFFEE,
    ICED_COFFEE,
    HOT_TEA,
    ICED_TEA,
    LEMONADE,
    GENERIC, // cola, water...
    MILKSHAKE,
    VIRGIN_COCKTAIL, // alcohol-free
    COCKTAIL; // with alcohol

}
```

The elemental classes of the project are shown above. The Food class includes the main attributes and the Drink class inherits them except kind. toString() functions are overridden to create the intended design of UI in the console screen.

```
public class FoodTree {

    public FoodTreeNode root;
    private int nodeCount = 0;

    public FoodTree()
    {
        root = null;
    }

    public void insert(Food food) {
        root = insert(food, root);
        setNodeCount((nodeCount+1));
    }

    private static FoodTreeNode insert(Food food, FoodTreeNode t) {
        if (t == null)
            t = new FoodTreeNode(food, null, null);
        else if (food.name.compareTo(t.getFood().name) < 0)
            t.leftChild = insert(food, t.leftChild);
        else if (food.name.compareTo(t.getFood().name) > 0)
            t.rightChild = insert(food, t.rightChild);
        else
            ;
        return t;
    }

    public FoodTreeNode search(Food food) {
        return search(food, root);
    }

    private static FoodTreeNode search(Food food, FoodTreeNode t) {
        if (t == null)
            return null;
        else if (food.name.toLowerCase().compareTo(t.getFood().name.toLowerCase()) < 0)
            return search(food, t.leftChild);
        else if (food.name.toLowerCase().compareTo(t.getFood().name.toLowerCase()) > 0)
            return search(food, t.rightChild);
        else {
            return t;
        }

    }

    public void remove(Food food) {
        root = remove(food, root);
        setNodeCount((nodeCount-1));

    }

    private static FoodTreeNode remove(Food food, FoodTreeNode t) {
        if (t == null)
            return t;
        if (food.name.compareTo(t.getFood().name) < 0)
            t.leftChild = remove(food, t.leftChild);
        else if (food.name.compareTo(t.getFood().name) > 0)
            t.rightChild = remove(food, t.rightChild);
        else if (t.leftChild != null && t.rightChild != null) {
            t.setFood(findMax(t.leftChild).getFood());
            t.leftChild = remove(t.getFood(), t.leftChild);

        } else if (t.leftChild != null)
            t = t.leftChild;
        else
            t = t.rightChild;
        return t;
    }
```

```
public class FoodTreeNode {

    private Food food;
    public FoodTreeNode leftChild, rightChild;

    FoodTreeNode(Food food, FoodTreeNode leftChild, FoodTreeNode rightChild) {
        this.food = food;
        this.leftChild = leftChild;
        this.rightChild = rightChild;
    }

    public Food getFood() {
        return food;
    }

    public void setFood(Food food) {
        this.food = food;
    }


}
```

```
public static FoodTreeNode findMax(FoodTreeNode root) {

    FoodTreeNode current = root;
    while(current.rightChild != null) {
        current = current.rightChild;
    }
    return current;
}

public int getNodeCount() {
    return nodeCount;
}

public void setNodeCount(int nodeCount) {
    this.nodeCount = nodeCount;
}

public void multipleInsert(Food[] foods) {
    for (int i = 0; i < foods.length; i++) {
        insert(foods[i]);
    }
}
```

Above are classes related to the food binary search tree. A tree class with its node class is created, and methods related to adding, removing, and searching Food nodes are implemented.

```
public class DrinkTree {

    public DrinkTreeNode root;
    private int nodeCount = 0;

    public DrinkTree()
    {
        root = null;
    }

    public void insert(Drink drink) {
        root = insert(drink, root);
        setNodeCount((nodeCount+1));
    }

    private static DrinkTreeNode insert(Drink drink, DrinkTreeNode t) {
        if (t == null)
            t = new DrinkTreeNode(drink, null, null);
        else if (drink.name.compareTo(t.getDrink().name) < 0)
            t.leftChild = insert(drink, t.leftChild);
        else if (drink.name.compareTo(t.getDrink().name) > 0)
            t.rightChild = insert(drink, t.rightChild);
        else
            ;
        return t;
    }

    public DrinkTreeNode search(Drink drink) {
        return search(drink, root);
    }

    private static DrinkTreeNode search(Drink drink, DrinkTreeNode t) {
        if (t == null)
            return null;
        else if (drink.name.toLowerCase().compareTo(t.getDrink().name.toLowerCase()) < 0)
            return search(drink, t.leftChild);
        else if (drink.name.toLowerCase().compareTo(t.getDrink().name.toLowerCase()) > 0)
            return search(drink, t.rightChild);
        else {
            return t;
        }

    }
    public void remove(Drink drink) {
        root = remove(drink, root);
        setNodeCount((nodeCount-1));

    }

    private static DrinkTreeNode remove(Drink drink, DrinkTreeNode t) {
        if (t == null)
            return t;
        if (drink.name.compareTo(t.getDrink().name) < 0)
            t.leftChild = remove(drink, t.leftChild);
        else if (drink.name.compareTo(t.getDrink().name) > 0)
            t.rightChild = remove(drink, t.rightChild);
        else if (t.leftChild != null && t.rightChild != null) {
            t.setDrink(findMax(t.leftChild).getDrink());
            t.leftChild = remove(t.getDrink(), t.leftChild);

        } else if (t.leftChild != null)
            t = t.leftChild;
        else
            t = t.rightChild;
        return t;
    }
```

```
public class DrinkTreeNode {

    private Drink drink;
    public DrinkTreeNode leftChild, rightChild;

    DrinkTreeNode(Drink drink, DrinkTreeNode leftChild, DrinkTreeNode rightChild) {
        this.drink = drink;
        this.leftChild = leftChild;
        this.rightChild = rightChild;
    }

    public Drink getDrink() {
        return drink;
    }

    public void setDrink(Drink drink) {
        this.drink = drink;
    }


}
```

```
public static DrinkTreeNode findMax(DrinkTreeNode root) {

    DrinkTreeNode current = root;
    while(current.rightChild != null) {
        current = current.rightChild;
    }
    return current;
}

public int getNodeCount() {
    return nodeCount;
}

public void setNodeCount(int nodeCount) {
    this.nodeCount = nodeCount;
}

public void multipleInsert(Drink[] drinks) {
    for (int i = 0; i < drinks.length; i++) {
        insert(drinks[i]);
    }
}
```

Above are classes related to the drink binary search tree. A tree class with its node class is created, and methods related to adding, removing, and searching Drink nodes are implemented.

```java
import java.util.ArrayList;

public class Menu {

    // Text coloring
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String ANSI_BOLD = "\u001b[1m";

    public ArrayList<Food>[] foodMenu = new ArrayList[11]; // food kinds = array size
    public ArrayList<Drink>[] drinkMenu = new ArrayList[9]; // drink kinds = array size


    // default constructor
    public Menu() {

    }

    // create menu
    public void printMenu(FoodTree foodTree, DrinkTree drinkTree) {

        //NPE-generation to prevent errors
        for (int i = 0; i < foodMenu.length; i++) {
            foodMenu[i] = new ArrayList<Food>();
        }

        //NPE-generation to prevent errors
        for (int i = 0; i < drinkMenu.length; i++) {
            drinkMenu[i] = new ArrayList<Drink>();
        }

        // traverse both trees, add each item to ArrayLists
        preorderFoodTree(foodTree.root);
        preorderDrinkTree(drinkTree.root);

        System.out.println(ANSI_BOLD + "FOODS" + ANSI_RESET);

        for (int i = 0; i < foodMenu.length; i++) {

            if(FoodKind.values()[i].name().contains("_")) {
                System.out.println("\t" + FoodKind.values()[i].name().replace("_"," "));
            }else {
                System.out.println("\t" + FoodKind.values()[i]);
            }

            for (int j = 0; j < foodMenu[i].size(); j++) {

                System.out.println("\t\t- " + foodMenu[i].get(j).toString());
            }
        }

        System.out.println(ANSI_BOLD + "\nDRINKS" + ANSI_RESET);

        for (int i = 0; i < drinkMenu.length; i++) {

            if(DrinkKind.values()[i].name().contains("_")) {
                System.out.println("\t" + DrinkKind.values()[i].name().replace("_"," "));
            }else {
                System.out.println("\t" + DrinkKind.values()[i]);
            }

            for (int j = 0; j < drinkMenu[i].size(); j++) {

                System.out.println("\t\t- " + drinkMenu[i].get(j).toString());
            }
        }

    }
```

```java
// Recursive tree traversal
public void preorderFoodTree(FoodTreeNode t) { // send tree root
    // root - left - right
    if (t == null)
        return;

    addMenu(t.getFood(), null);

    preorderFoodTree(t.leftChild);
    preorderFoodTree(t.rightChild);

}

// Recursive tree traversal
public void preorderDrinkTree(DrinkTreeNode t) { // send tree ro
    // root - left - right
    if (t == null)
        return;

    addMenu(null, t.getDrink());

    preorderDrinkTree(t.leftChild);
    preorderDrinkTree(t.rightChild);

}

public void addMenu(Food food, Drink drink) {

    if (food != null) {
        int foodIndex = food.getFoodKind().ordinal();
        foodMenu[foodIndex].add(food);
    } else {
        int drinkIndex = drink.getDrinkKind().ordinal();
        drinkMenu[drinkIndex].add(drink);
    }

}
```

The Menu class is shown above. Two arrays of ArrayLists are defined with the number of constants of enumerations “FoodKind” and “DrinkKind” where the array index corresponds to enumeration ordinals. It is possible to print the menu by traversing each tree and adding items to their related kinds.

```java
import java.util.Scanner;
import java.text.DecimalFormat;

public class Main {

    public static void main(String[] args) {

        // Begin Populate Menu
        Menu menu = new Menu(); // create menu
        FoodTree foods = new FoodTree(); // create food tree
        DrinkTree drinks = new DrinkTree(); // create drink tree

        // --- FOODS ---

        // --Breakfast--
        Food f1 = new Food("Panini Toast", new IngredientsLinkedList(new String[] { "Tomatoes", "White Cheese" }),
                FoodKind.BREAKFAST,
                new ContentWarningLinkedList(
                        new ContentWarnings[] { ContentWarnings.GLUTEN, ContentWarnings.SOY, ContentWarnings.LACTOSE }),
                5.99);

        Food f2 = new Food("Menemen", new IngredientsLinkedList(new String[] { "Tomato", "Pepper", "Eggs" }),
                FoodKind.BREAKFAST,
                new ContentWarningLinkedList(new ContentWarnings[] { ContentWarnings.LACTOSE, ContentWarnings.EGG }),
                9.99);
```

```java
Drink d24 = new Drink("Long Island Ice Tea",
        new IngredientsLinkedList(new String[] { "Vodka", "Gin", "Rum", "Tequila", "Curaçao",
                "Sweet & Sour Mix", "Coca Cola Classic / Fruity" }),
        DrinkKind.COCKTAIL, new ContentWarningLinkedList(new ContentWarnings[] {}), 29.99);

Drink d25 = new Drink("Health Potion +100 HP",
        new IngredientsLinkedList(
                new String[] { "Mappy Hoons Special Electrons", "Refreshing Red Fruit Syrup" }),
        DrinkKind.COCKTAIL, new ContentWarningLinkedList(new ContentWarnings[] {}), 999.99);

foods.multipleInsert(new Food[] { f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15, f16, f17,
        f18, f19, f20, f21, f22, f23, f24, f25, f26, f27, f28, f29, f30, f31, f32, f33 });
drinks.multipleInsert(new Drink[] { d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15, d16, d17,
        d18, d19, d20, d21, d22, d23, d24, d25 });
// End Populate Menu

// User Prompts
System.out.println("\n");
Scanner in = new Scanner(System.in); // begin Scanner
System.out.println("Welcome to Mappy Hoons !\n");
System.out.println("Choose action \n1-Display Menu\n2-Search Food/Drink\n3-Place Order\n4-Exit");

int menuSelection = 0;

while (menuSelection != 4) {
    menuSelection = in.nextInt();
    in.nextLine(); // reset Scanner after int input

    if (menuSelection == 1) { // display menu
        menu.printMenu(foods, drinks);

    } else if (menuSelection == 2) {

        System.out.print("Enter item name: ");
        String itemName = in.nextLine();
        Food temp = new Food(itemName, null, null, null, 0.0);
        Drink temp2 = new Drink(itemName, null, null, null, 0.0);

        if (foods.search(temp) != null) {
            temp = foods.search(temp).getFood();
            System.out.println("Item found : " + temp.toString());

        } else if (drinks.search(temp2) != null) {
            temp2 = drinks.search(temp2).getDrink();
            System.out.println("Item found : " + temp2.toString());

        } else {
            System.out.println("Item not found !");
        }

    } else if (menuSelection == 3) {

        double sum = 0.0;
        String order = "";

        while (!(order.toLowerCase().equals("order"))) {

            System.out.print("Enter item to order, to finish ordering, type \"order\" : ");
            order = in.nextLine();
            order = order.toLowerCase();

            Food temp = new Food(order, null, null, null, 0.0);
            Drink temp2 = new Drink(order, null, null, null, 0.0);

            if (foods.search(temp) != null) {
                temp = foods.search(temp).getFood();
                System.out.println("Item found : " + temp.toString());
                sum += temp.cost;

            } else if (drinks.search(temp2) != null) {
                temp2 = drinks.search(temp2).getDrink();
                System.out.println("Item found : " + temp2.toString());
                sum += temp2.cost;

            } else {
                if (!(order.equals("order"))) {
                    System.out.println("Item not found !");
                }
            }

        }
        DecimalFormat df = new DecimalFormat("0.00");
        System.out.println("Order completed. Total cost is $" + df.format(sum));

    } else if (menuSelection == 4) { // terminate program
        System.exit(0);

    }else {
        System.out.println("Invalid option !");

    }

    if (menuSelection != 4) {
        System.out.println("\nChoose action \n1-Display Menu\n2-Search Food/Drink\n3-Place Order\n4-Exit");
    }
}
```

```
Welcome to Mappy Hoons !

Choose action
1-Display Menu
2-Search Food/Drink
3-Place Order
4-Exit
1
FOODS
        BREAKFAST
                - Panini Toast $5.99
                        Ingredients: Tomatoes, White Cheese
                        Content Warnings: GLUTEN, SOY, LACTOSE
                - Menemen $9.99
                        Ingredients: Tomato, Pepper, Eggs
                        Content Warnings: LACTOSE, EGG
                - Mushroom Omelette $12.0
                        Ingredients: Three Eggs, Mushrooms
                        Content Warnings: SOY, EGG
        STARTER
                - Chicken in London $16.99
                        Ingredients: Chicken Strips, Cajun Spices, French Fries, Honey Mustard Sauce
                        Content Warnings: GLUTEN, EGG, LACTOSE, MUSTARD, CELERY
                - Mac and Cheese $18.0
                        Ingredients: Elbows Macaroni, Cheese Sauce, Jalapeno Peppers, Mexican Beans, Spicy Bolognese Sauce, Frie
                        Content Warnings: GLUTEN, EGG, LACTOSE
                - Onion Rings $6.49
                        Ingredients: Ten Onion Rings, Asianda Sauce
                        Content Warnings: EGG
        SALAD
                - Caesar Salad $11.99
```

```
Welcome to Mappy Hoons !

Choose action
1-Display Menu
2-Search Food/Drink
3-Place Order
4-Exit
2
Enter item name: Chicken in london
Item found : Chicken in London $16.99
                        Ingredients: Chicken Strips, Cajun Spices, French Fries, Honey Mustard Sauce
                        Content Warnings: GLUTEN, EGG, LACTOSE, MUSTARD, CELERY

Choose action
1-Display Menu
2-Search Food/Drink
3-Place Order
4-Exit
3
Enter item to order, to finish ordering, type "order" : chicken in london
Item found : Chicken in London $16.99
                        Ingredients: Chicken Strips, Cajun Spices, French Fries, Honey Mustard Sauce
                        Content Warnings: GLUTEN, EGG, LACTOSE, MUSTARD, CELERY
Enter item to order, to finish ordering, type "order" : passion lemonade
Item found : Passion Lemonade $4.99
                        Ingredients: Lemon, Passion Fruit, Ice)
Enter item to order, to finish ordering, type "order" : ORDER
Order completed. Total cost is $21.98
```

The Main class and the representation of the main function, the console screen, are shown above. The menu items are populated regarding their trees and an infinite while loop of action menu is created, which includes the options to display the menu, search for food/drink, place an order, or exit (terminates the application).

4. Conclusion / Summary

Some methods such as “remove(item)” were implemented in tree classes to grant additional functionality, but weren’t used significantly. Also, hidden “admin” commands were introduced and coded to some extent during the project, to add new menu items or remove existing ones while the program was running. This idea, however, wasn’t fully integrated so it was left for future development.

Overall, this project successfully achieved its goal to provide a working restaurant menu with several possible interactions and a decent-looking UI. It is easy to provide user inputs and the application gives enough visual feedback to guide the user. No additional data structures than previously-stated ones were used during implementation, thanks to careful planning and execution. At all phases of the project, thorough research was conducted to gather information on how the application would function and how the menu would be populated.

5. References

We used course materials from SEN2211, SEN2212, SEN1002 & SEN2022.
We made additional research on the websites “GeeksforGeeks” and “W3Schools”