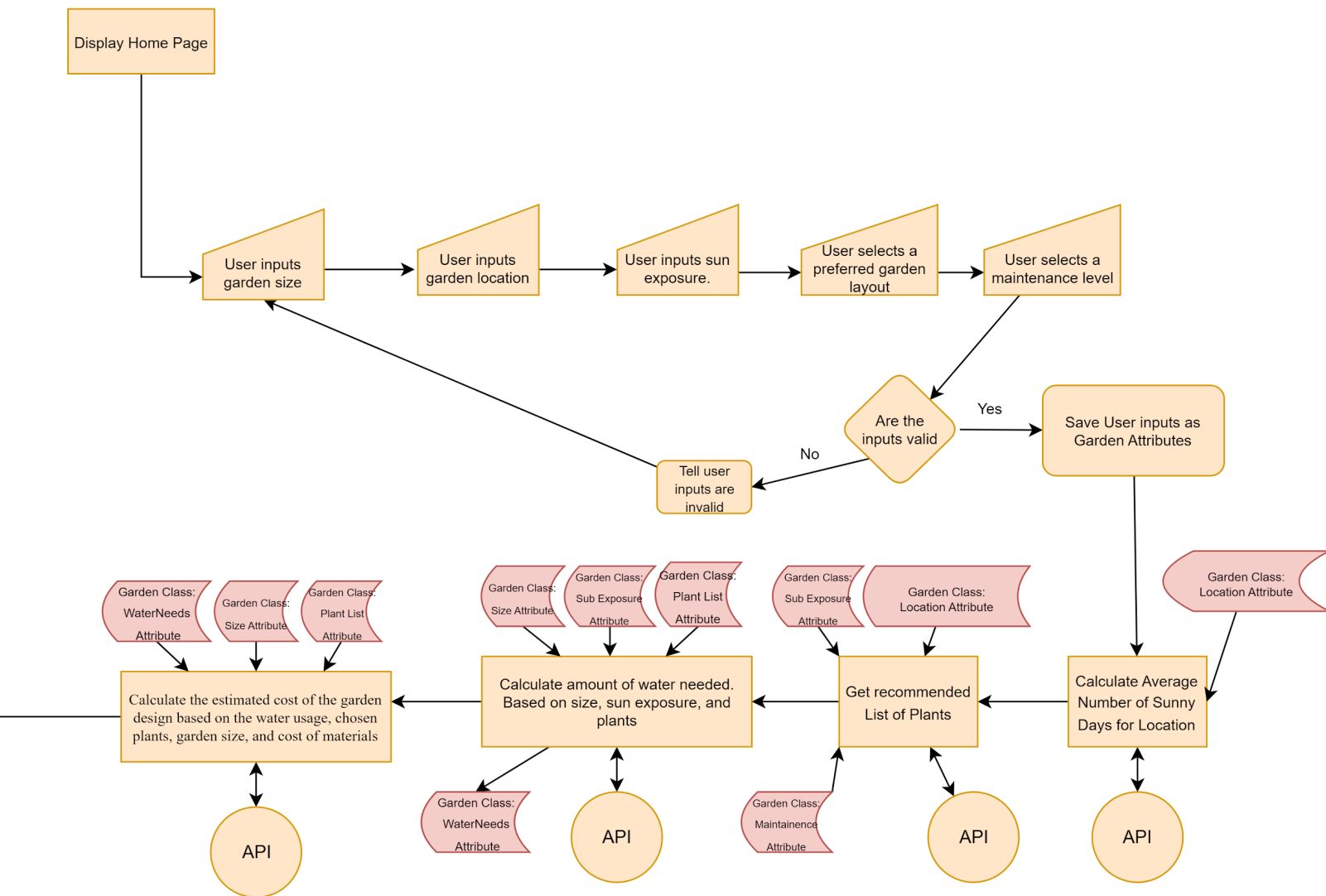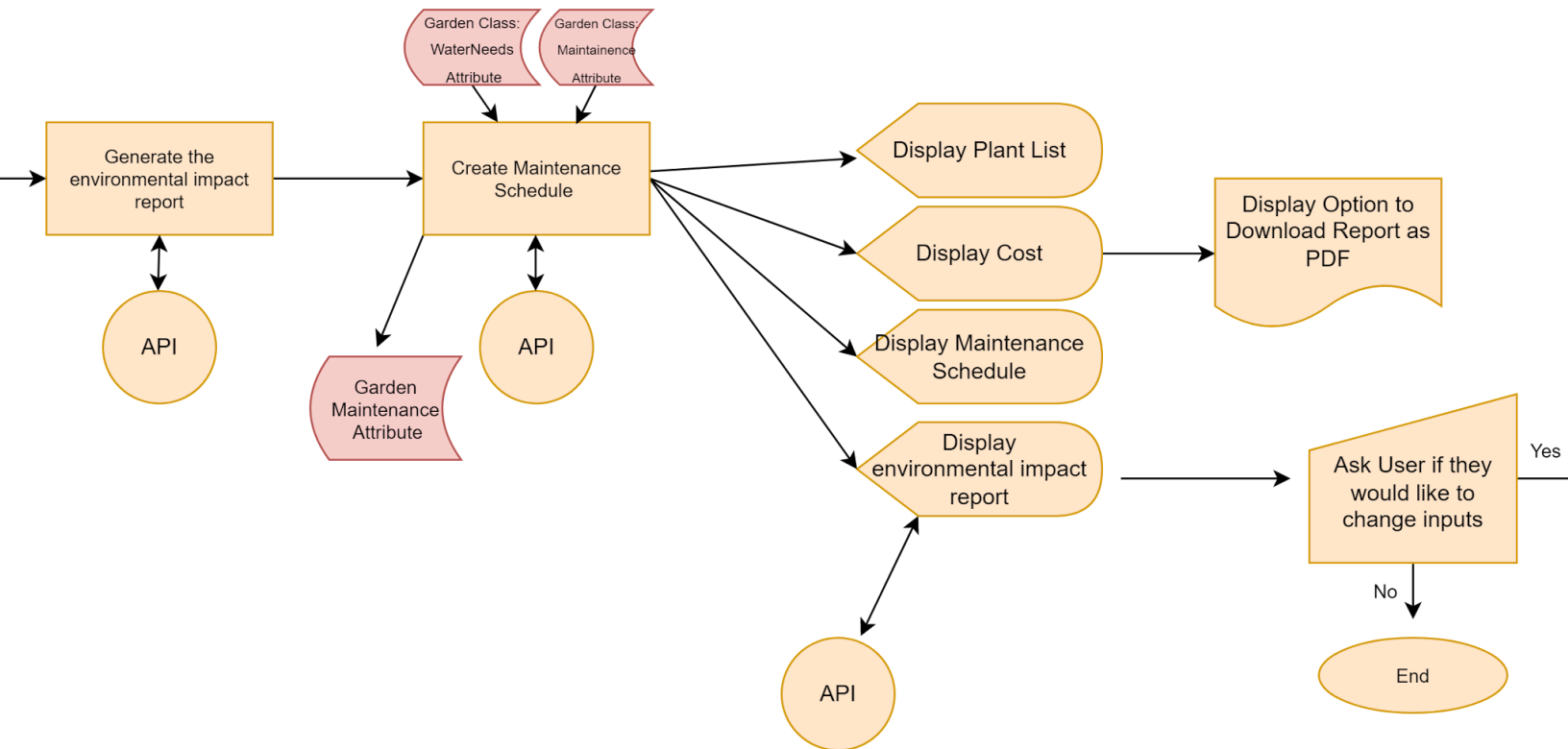Rohan Muppa
Ms. Nishiwaki
IB Computer Science SL
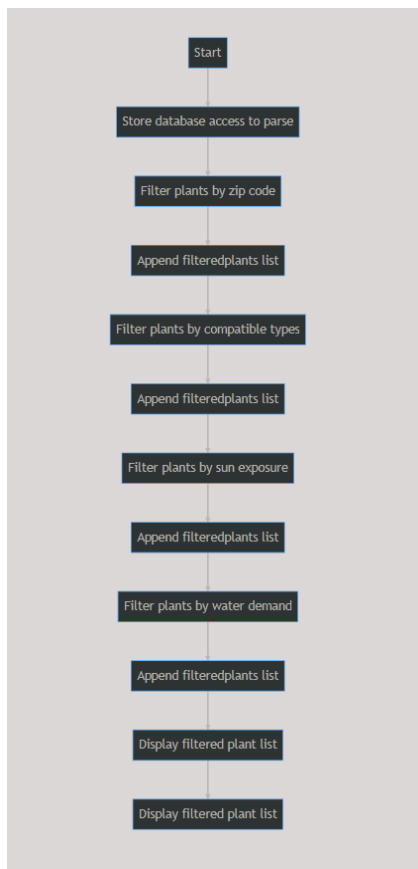March 20, 2023
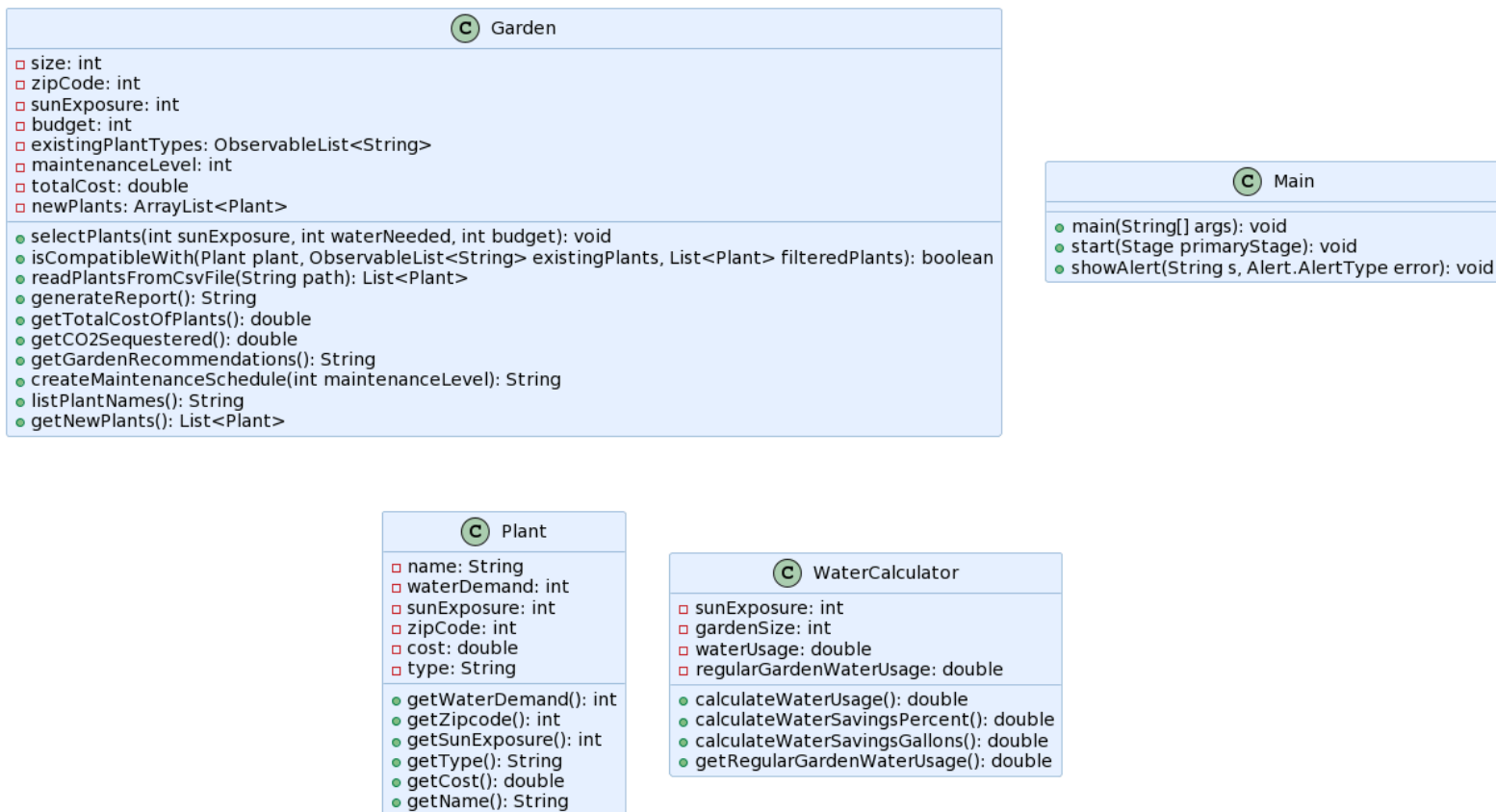
Criterion B:

Flowchart:

Z

Algorithms:

The main algorithm I had to incorporate was the algorithm that was used to select plants. This was used to filter all the plants from a database to fit the criteria of the user.

Data Structures:

I used ArrayLists to store the list of new plants ObservableLists were used to keep track of the types of plants that the user inputted and to dynamically update the UI. HashMaps were used to store information about each plant, with each plant's name serving as the key and its information, such as compatible plant types and their total count in the garden, stored as values.
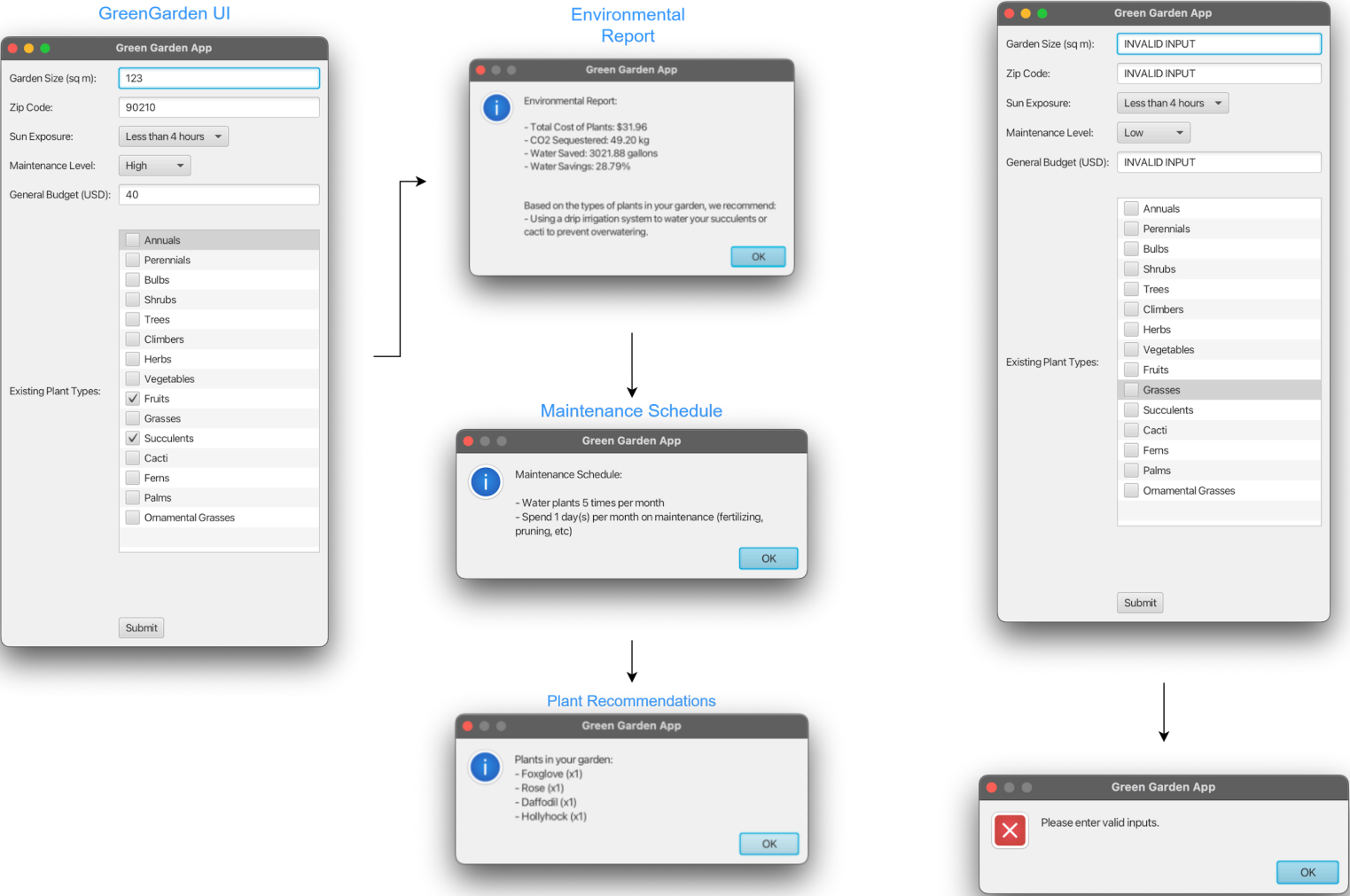
UML Diagram:

Garden

- size: int
- zipCode: int
- sunExposure: int
- budget: int
- existingPlantTypes: ObservableList<String>
- maintenanceLevel: int
- totalCost: double
- newPlants: ArrayList<Plant>

- selectPlants(int sunExposure, int waterNeeded, int budget): void
- isCompatibleWith(Plant plant, ObservableList<String> existingPlants, List<Plant> filteredPlants): boolean
- readPlantsFromCsvFile(String path): List<Plant>
- generateReport(): String
- getTotalCostOfPlants(): double
- getCO2Sequestered(): double
- getGardenRecommendations(): String
- createMaintenanceSchedule(int maintenanceLevel): String
- listPlantNames(): String
- getNewPlants(): List<Plant>

Main

- main(String[] args): void
- start(Stage primaryStage): void
- showAlert(String s, Alert.AlertType error): void

Plant

- name: String
- waterDemand: int
- sunExposure: int
- zipCode: int
- cost: double
- type: String

- getWaterDemand(): int
- getZipcode(): int
- getSunExposure(): int
- getType(): String
- getCost(): double
- getName(): String

WaterCalculator

- sunExposure: int
- gardenSize: int
- waterUsage: double
- regularGardenWaterUsage: double

- calculateWaterUsage(): double
- calculateWaterSavingsPercent(): double
- calculateWaterSavingsGallons(): double
- getRegularGardenWaterUsage(): double

Test Plan:

Success Criteria	Test
The program should be able to handle a multiple inputs, including different levels of maintenance (low, moderate, high) and amounts of sunlight exposure (less than 4 hours, 4-6 hours, 6-8 hours, more than 8 hours)	Testing various combinations of inputs
The program should be able to generate at least 1 plant for each garden	Test with multiple garden plans, comparing the program-generated plants to the actual plants
The program should be able to generate a report that shows the environmental standing of the garden, including the amount of CO_2 sequestered, the amount of water saved	Compare the program's reports with actual environmental data
The program should be able to recommend specific tips based on the specific needs of the plants	Test with various plant inputs. Ensure that tips are relevant, reasonable, and helpful
The garden and recommendations should predict a reduction in water usage by at least 10% compared to traditional gardens and gardening practices.	Compare water usage data from traditional gardening practices
The program should be able to recommend at least one plant option that is native to the specified	Test with multiple locations to test the

location to promote biodiversity and support local ecosystems	program's recommendations
Must provide users with a schedule for watering, fertilizing, and pruning	Test alerts with various settings
The total cost of the plants should be less than the budget	Test outputs using various budget inputs to make sure the total cost is always under the budget

UI Flow:

GreenGarden UI

Environmental Report

Maintenance Schedule

Plant Recommendations

Data Validation

Word Count: 112