Design Overview for Garden Game

High Distinction Level

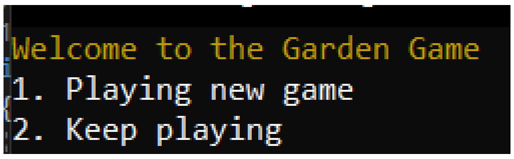
Name: Le Nho Bach

Student ID: 103487884

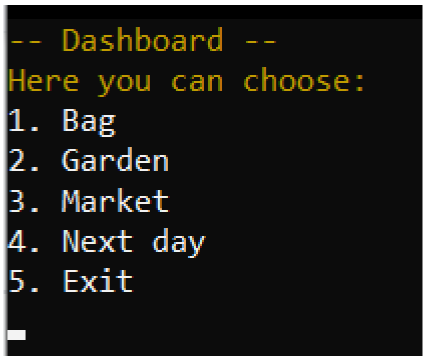
Summary of Program

This program is a game that allows players to build their own gardens. Players can buy and sell resources like seeds, water or fertilizer, and buy the plant holder like pots or normal plots to raise flowers or fruits. They can also sell the plant after it is fully-growth, or sell the plant holder if players feel like their garden is tight.

In this version, the game automatically saves the data after players quit, so they can still keep playing if they want.



```
Welcome to the Garden Game
1. Playing new game
2. Keep playing
```



```
-- Dashboard --
Here you can choose:
1. Bag
2. Garden
3. Market
4. Next day
5. Exit
```

The game now has colours, which allows the player to see the command, their choice and other information about the game more clearly. Despite being a small change, this gives players much better performance compared to the Distinction level program.

The program uses OOP Design, with multiple principles used to increase the performance and clarity of the code.

Required Roles

Class Bag

Responsibility	Type Details	Notes
AddSeed	void method	Add a seed to the bag
UseSeed	void method	Remove the seed from the bag and add it to a plant holder
UseWater	void method	Lose an amount of water in bag
Use Fertilizer	void method	Lose an amount of fertilizer in the bag
AllSeedTypeOf	bool method	Check whether a type of seed is in the bag or not
Money	double property	Return and change the money in the bag
NumberOfSeed	read-only int property	Return the number of seeds in the bag
Fertilizer	double property	Return and change the amount of fertilizer in the bag
Water	double property	Return and change the amount of water in the bag

Class PlantHolder

Responsibility	Type Details	Notes
AddSeed	void method	Add a seed to an empty plant holder
OneDayPass	void method	Update the plant after a day

CheckDone	virtual void method	Update the price of the plant holder after the plant is fully-growth
CheckWatered	void method	Check whether the plant is watered enough that day or not
Dead	void method	Make the plant dies due to the lack amount of water
SeedType	WaterDemand property	Return the type of the seed
Price	double property	Return the price of the plant holder
DaysRemaining	int property	Return how many days left until the plant is fully-growth
ShortDescription	virtual string property	Return a short description of the plant holder
LongDescription	string property	Return a long description of the plant holder
Seed	Seed property	Return or change the seed in the plant holder
Seeded	boolean property	Return whether the plant holder has a seed inside or not
HolderType	virtual HolderType property	Return the type of the plant holder
Name	string property	Return the name of the plant holder
WaterToday	double property	Return the amount of water has been watered to the plant today
Fully_Growth	boolean property	Return whether the plant is fully-growth or not

Class FlowerPot: A child class of the class PlantHolder

Responsibility	Type Details	Notes
HolderType	override HolderType property	Return the type of the plant holder

Class Plot: A child class of the class PlantHolder

Responsibility	Type Details	Notes
HolderType	override HolderType property	Return the type of the plant holder

Class Garden

Responsibility	Type Details	Notes
AddPlantHolder	void method	Add a plant holder to the garden
RemovePlantHolder	void method	Remove a plant holder from the garden
OneDayPass	void method	Update all the plant holder in the garden after a day pass
Sell	void method	Remove/Reset /sell the plant
ResetPlantHolder	void method	Reset a plant holder
HolderList	List<PlantHolder> property	Return the list of all plan holders in the garden
Description	string property	Return the description of the garden

Struct Seed

Responsibility	Type Details	Notes
Description	string property	Return the information of the seed

Class BuyingState

Responsibility	Type Details	Notes
SeedShopDescription	string property	Return the description of all seed in the shop
FlowerPotShopDescription	string property	Return the description of all pot in the shop
SeedCollection	List<Seed> property	Return the collection of seeds
PotCollection	List<FlowerPot> property	Return the collection of flower pots

Class System

Responsibility	Type Details	Notes
GetIntBetween	int function	Get the max and the min integer value, then assure that the user would give the valid input

GetDoubleBetween	double function	Get the max and the min double value, then assure that the user would give the valid input
AskIntAmount	int function	Assure the user to give the valid amount of something
AskDoubleAmount	double function	Assure the user to give the valid amount of something
PressEnter()	void function	Let user click a button before clear the screen
CheckMoney	bool function	Test whether the user can buy something
RestoreData	**void function**	**Load the data from a text**
LoadSeed	**Seed function**	**Return the seed based on its saved name**
SaveData	**void function**	**Save the current gameplay**
Dashboard	void function	Show the dashboard of the game
BagBoard	void function	Show the bag of the player

GardenBoard	void function	Show the garden of the player
VisitPlantHolder	void function	Allow player to interact with the plant holders in the garden
BuyingPlantHolder	void function	Allow player to buy new plant holders
BuyingSeed	void function	Allow player to buy seeds
BuyingWater	void function	Allow player to buy water
Buying Fertilizer	void function	Allow player to buy fertilizer
VisitSellingState	void function	Allow player to sell plant holders
Start	void function	Start the game

(New features are bold)

Enum Type

Value	Notes
None	Nothing
Flower	This is a flower seed and can only be planted in a flower pot
Fruit	This is a fruit seed and can only be planted in a plot

Enum WaterDemand

Value	Notes
None	Nothing
NeedWater	This seed need to be watered
DontNeedWater	This seed does not need to be watered

Enum HolderType

Value	Notes
Plot	This is used to plant fruit
FlowerPot	This is used to plant flower

The use of abstraction: GetIntBetween(int start, int end)

This is one of the most useful functions in my whole program. For details, it gets 2 integer values: start and end. Then the program will ask the user to give the valid input, which is an integer between start and end values. After creating this function, I do not need to care about how will I require the user to give a choice and assure that the number is valid anymore

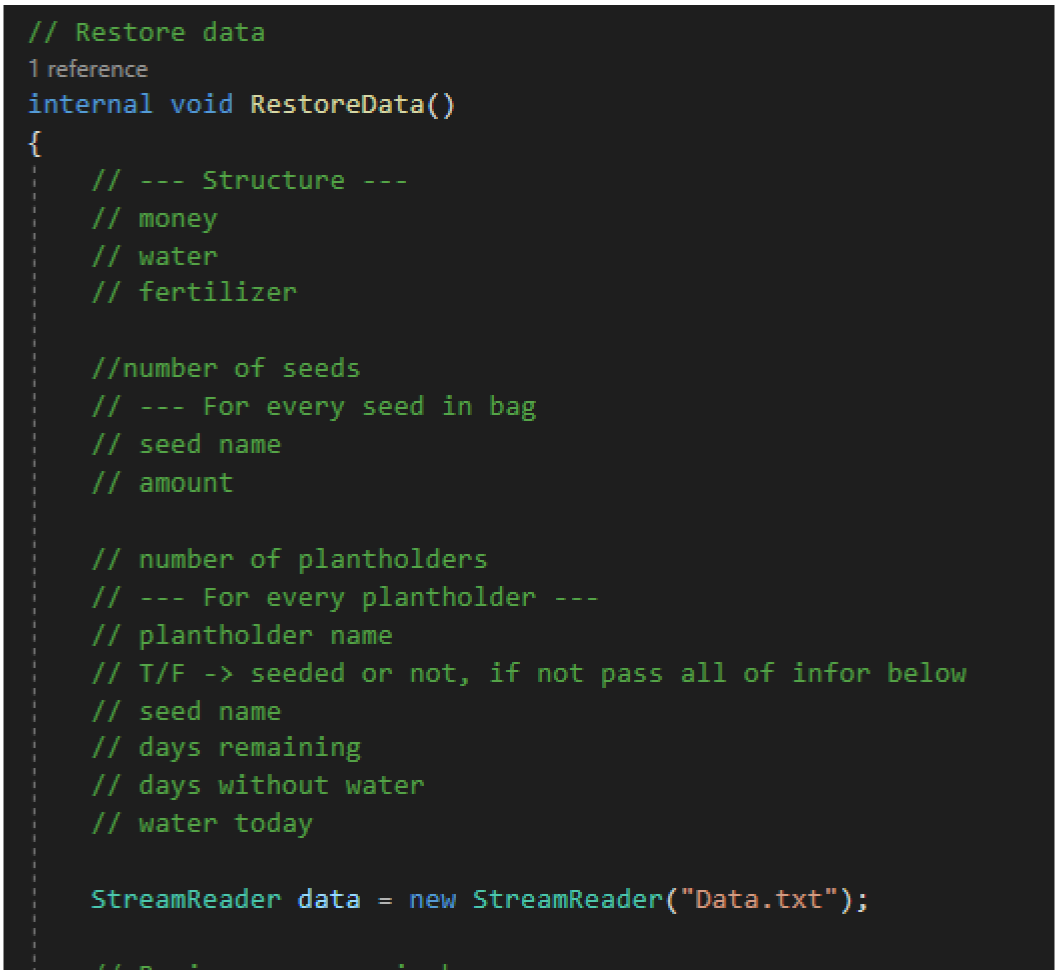
The use of inheritance and polymorphism:

The relationship between classes PlantHolder, Plot and FlowerPot.

The FlowerPot and Plot inherit the features, functions, properties... of the class PlantHolder. However, when we call HolderType, the FlowerPot will return HolderType.FlowerPot while the Plot will return the HolderType.Plot

Additional features and complexity:

After I asked my lecturer about available upgradation for this program to get an HD for the whole course, he said that it would be great if players can save their game progress, so they do not need to start the game again. I think that is a good idea and this is how I save the data:



```
// Restore data
1 reference
internal void RestoreData()
{
    // --- Structure ---
    // money
    // water
    // fertilizer

    //number of seeds
    // --- For every seed in bag
    // seed name
    // amount

    // number of plantholders
    // --- For every plantholder ---
    // plantholder name
    // T/F -> seeded or not, if not pass all of infor below
    // seed name
    // days remaining
    // days without water
    // water today

    StreamReader data = new StreamReader("Data.txt");
```

Serialization is not ideal for saving enums or dictionaries, so I simplified everything by using text (save by StreamWriter, and load by StreamReader). The data text includes

- Money, water, and fertilizer are saved as normal numbers
- The next line is the number (n) of seeds in the bag. The next n lines are the names and amounts of them. The seeds will be recognized by the program by a new prebuilt dictionary, which has seed names as keys and the Seed instances as values.
- The next line is the number m of plant holders and the next m lines are the information of the plant holders. There is also a new dictionary that has flower pots' names as keys and the Flower Pot instances as values. I will call each plant holder's information a 'block'. The first line is the name of the plant holder. If the name is "Plot", it is a plot instead of a flower pot. The next line is "T" or "F", representing whether that plant holder has a seed or not. It has if the letter is "T", and vice versa. If the plant holder does not have a seed, we read the next block. The four next lines contain the information of the plant in case the plant holder has had a seed already. The seed name will be translated

similarly to translating the seeds in the bag, which I have mentioned above. The days remaining until the plant is fully grown, the days without water or the amount of water we watered today will be read and parsed normally.

After the player quit the game, the data will be saved based on the above structure. The seed and the plant holder will be saved through their name.

Besides adding the data-saving features, I also add colours to the program using ConsoleColor. As a developer, I have played this game hundreds of times and saw that the interface is too simple and sometimes it is problematic to read new lines. Therefore, I added the colours and it extremely made the game better. Despite knowing how to use more colours, I think adding only dark yellow is enough and the most effective.

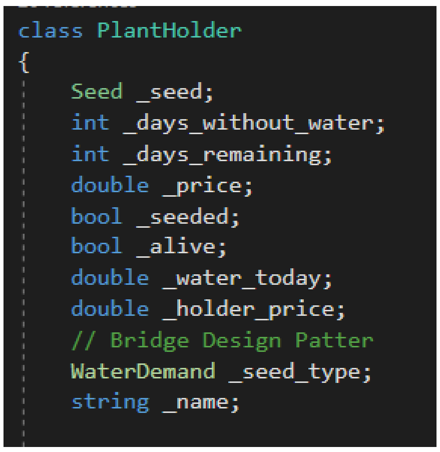
For these changes, I think that my program is at a complexity more than a D-level custom program.

Design Patterns:

I have used 3 design patterns in this program: Singleton, Template Method, and Bridge.

Bridge Design Pattern:

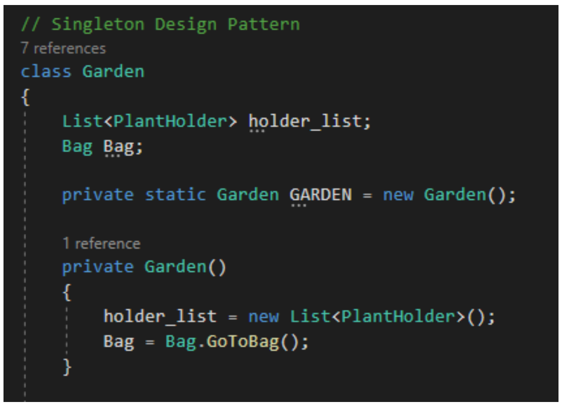
At the moment, Bridge does not bring too much benefit in improving my program. However, it makes the game clearer and can be developed more in the future. First, I want to specify whether the plant holder is a flower pot or plot, and does it need water. If I split them based on the characteristics, there will be four classes, which is not necessary because it can be hard to maintain and fix, while the functions are not that different. Then I found Bridge, which allows me to have only 2 child classes and keep 2 other characteristics as features. The Bridge Design Pattern helps me not to create more and more inherited classes.



```
class PlantHolder
{
    Seed _seed;
    int _days_without_water;
    int _days_remaining;
    double _price;
    bool _seeded;
    bool _alive;
    double _water_today;
    double _holder_price;
    // Bridge Design Patter
    WaterDemand _seed_type;
    string _name;
```

Singleton Design Pattern:

Singleton is a simple method, but it can help you improve your code significantly. There is only one bag and one garden in my game, but I want to access them from everywhere. Normally, if I want to do that, I would need to clone the instances of Garden and Bag many times, which makes my code more and more complicated and hard to fix. To solve this, I used Singleton. I created the instance inside the class, private the constructor, and then made another static method to access the created instance.



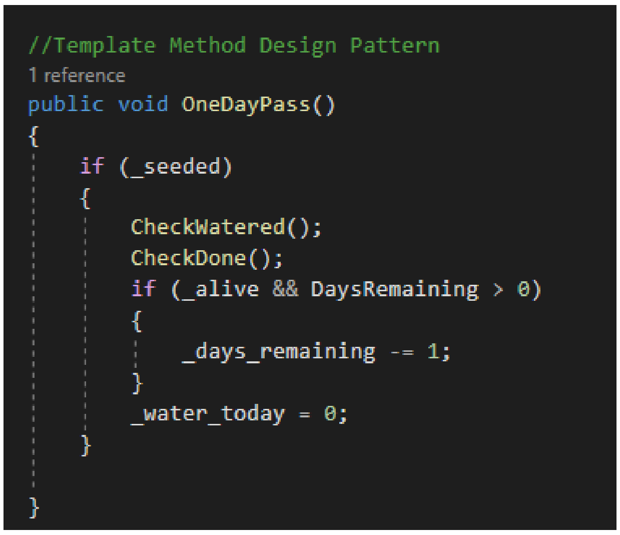
```
// Singleton Design Pattern
7 references
class Garden
{
    List<PlantHolder> holder_list;
    Bag Bag;

    private static Garden GARDEN = new Garden();

    1 reference
    private Garden()
    {
        holder_list = new List<PlantHolder>();
        Bag = Bag.GoToBag();
    }
```

Template Method Design Pattern:

As I mentioned above, there are two kinds of plant holders: Flower Pot and Plot. The price of a fully grown plant in a flower pot differs from the price of a fully grown plant in a plot. However, there are just a few different steps between them, so I decided to use Template Method. This design pattern is a process of methods. Plot and Garden may have the same process, but the results are still different due to differences in the methods used inside. For more details, the CheckDone() method calculated the price of the plant holder unlikely between Flower Pot and Plot. But I only need to change one method inside instead of creating more and more override methods. Template Method makes my code clearer and simpler.



```
//Template Method Design Pattern
1 reference
public void OneDayPass()
{
    if (_seeded)
    {
        CheckWatered();
        CheckDone();
        if (_alive && DaysRemaining > 0)
        {
            _days_remaining -= 1;
        }
        _water_today = 0;
    }

}
```