CSCI 1110 – Fall 2021

Assignment 03 – Due 28/11, 11 pm

Please start this assignment early; programming and logic take time - if you leave it to the last minute, you might not have enough time to finish or might make silly mistakes that you otherwise could avoid. Note that TAs and instructors will not be able to answer last-minute questions!

All work is to be handed in Mimir, our online code learning environment. You should, however, write your code on an IDE such as IntelliJ.

To complete this assignment, you will need to know about:

- Basic Java
- Conditionals
- Boolean Variables
- Simple Input Validation
- Loops
- Methods
- Basic Objects and Classes
- Instance Methods
- ArrayLists
- Inheritance (including the instanceof operator)
- Abstract classes

Your code **must compile**. If it does not compile, you will receive a 0 (zero) on that portion of the assignment, and no partial marks will be given.

Remember that students who ***hardcode*** their outputs to match the test cases in Mimir will receive a **zero** on the entire assignment.

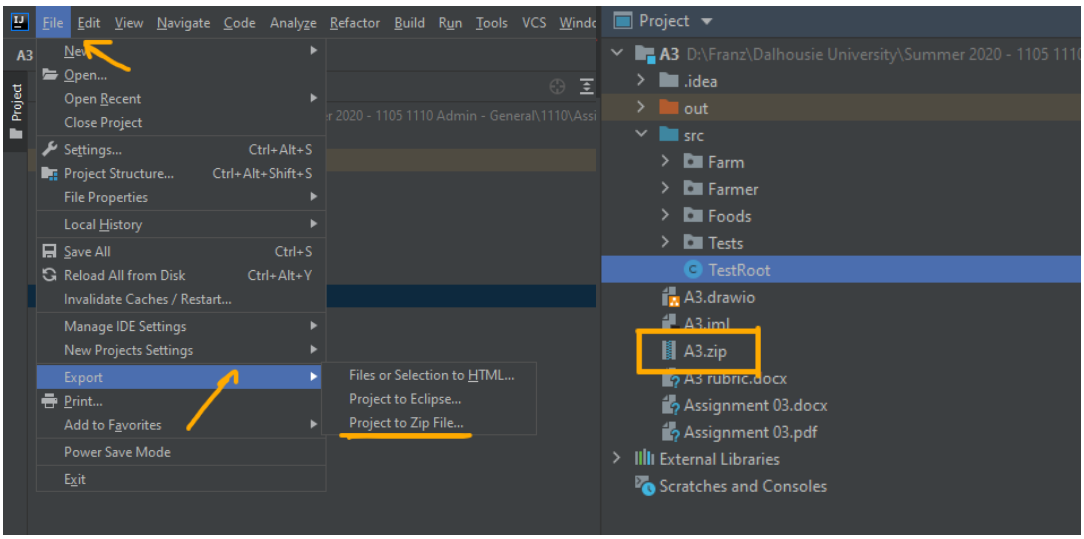
Grading Scheme: Please see the grading scheme at the end of this document.

Coding Style: You must provide proper variable names and comments. Please follow the guidelines on https://web.cs.dal.ca/~franz/CodingStyle.html

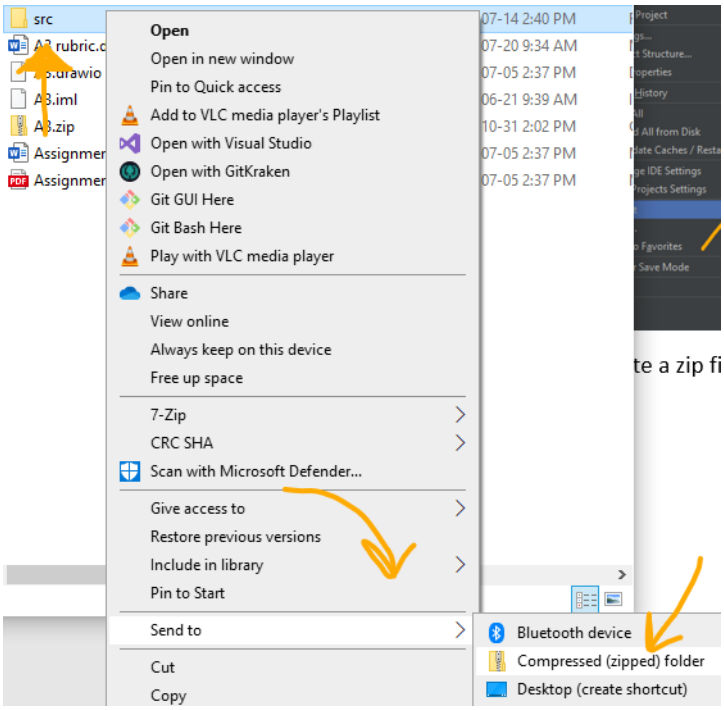
How to submit A3

Assignment 3 will use **packages**. You must use the provided package structure otherise the test cases will not compile. You **have two options** to upload your solution to mimir:

1. Export your project as a zip file in IntelliJ and upload the zip file



2. Create a zip file of your ***src*** filder



Assignment Overview: Farmer Joe

In this assignment, you will write classes to represent a Farm, the Farmer, and the Foods that the farmer can seed and harvest on the farm. A farm is an entity that has a fertile area and a specific type of soil. Foods (such as fruits and vegetables) require a certain farm space to grow and grow in any soil type. However, Foods seeded in their preferred Soil will grow faster.

This entire assignment will be unit tested (similar to A2).

Note: I know very, very little about farming. The goal of this assignment is not to master the art of agriculture and model it in Java. Instead, you should focus on practicing writing OOP code using Inheritance.

Note 2: This assignment is partially inspired by the game Stardew Valley. I've never played the game before, so don't take it too seriously 😊

The starter code is available in Brightspace and in Mimir. Please respect the package structure and submit your files in the same format. The starter code contains an Enum type (more info here if you want: https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html) to represent the various types of soil that a farm can have. If you wish, for example, to set the preferred soil of a given farm to Clay, you can:

```
Soil mySoil = Soil.Clay
```

You can compare two Soil variables directly with the == operator:

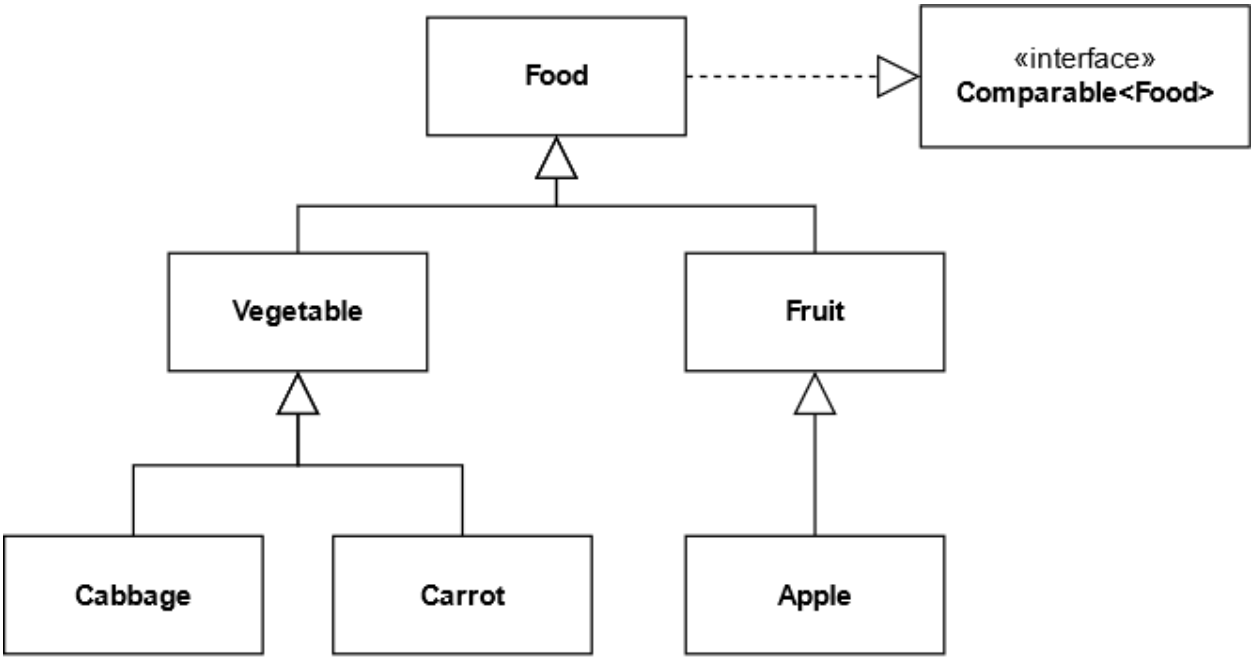
```
if(mySoil == Soil.Silt) // this will evaluate to false
```

I've also made available a zip file containing all test cases that run on mimir. If you want, you can put the files in your solution (**please do not upload those files to mimir after**) and run the main method to run all test cases.

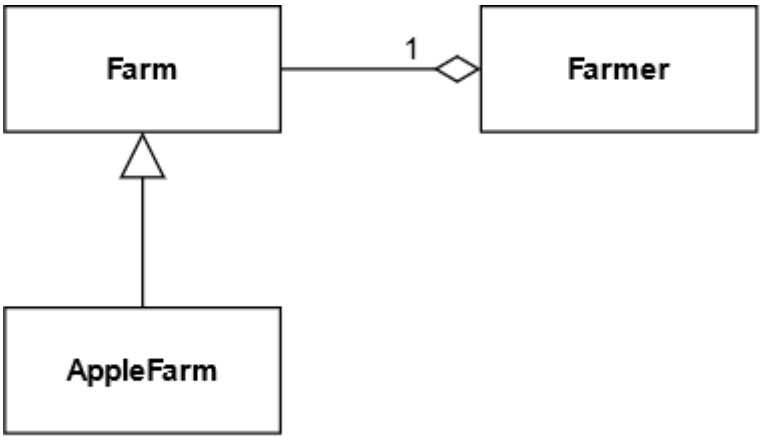
There are three packages in the starter code:

- **Foods**: All classes related to foods that can be seeded and harvested. This includes a generic abstract class like Food.java and very specialized classes like Apple.java
- **Farm**: All classes related to the farm, like Farm.java and Soil.java, belong in this package. Soil.java is fully implemented in the starter code
- **Farmer**: This package contains the Farmer class. Although I've been calling him Joe, you can name the farmer to your desire.

The Food Inheritance Tree



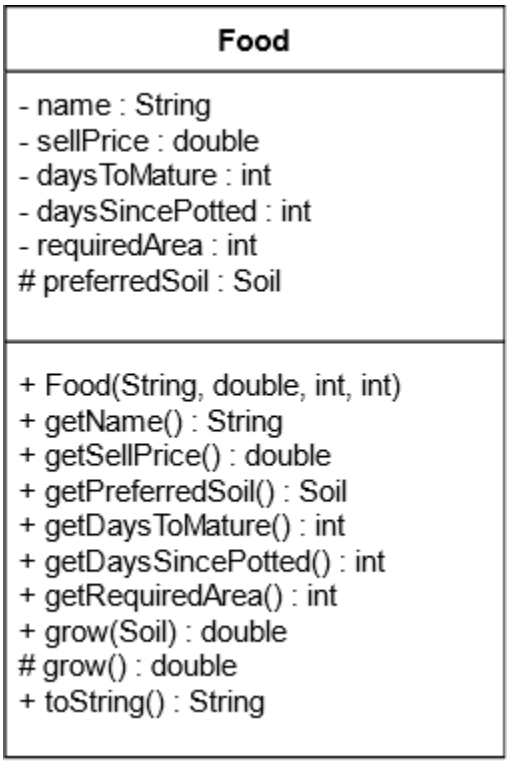
The Farm Inheritance Tree and Farmer Relationship



Food

As mentioned before, foods require a certain area to be seeded and have a Soil preference. **The food class is abstract**. Below is the UML diagram for Food. Please note that you can and might have to create extra **private variables and private methods** on the classes of this assignment.

The food class implements the comparable interface. Foods are ordered by their sellPrice (highest first). Remember that compareTo **must** return an int.



Methods:

- Food(String, double, int, int) : This constructor receives a food's name, selling price, how many days it needs to mature and its required area. Abstract foods don't have a preferred soil.
- grow(Soil) : is abstract
- grow() : This method can be called "every night" to add one more day to the food's growth cycle. It should return the percentage (between 0.00 and 1.00) of the food's growth cycle. If the food is potted for more than its required days to mature, this method should return 1.00.
- toString() : returns a formatted string according to the example:

```
                Cabbage (Clay) - 3/10 days
Food Name (Preferred Soil) - days potted / days to mature
```

Food Subclasses

The diagrams for the subclasses of food are below.

Vegetable	Fruit
+ Vegetable(String, double, int, int) + grow(Soil) : double	+ Fruit(String, double, int, int) + grow(Soil) : double

Vegetable and Fruit are **not** abstract. They will override the grow(Soil) method providing a concrete implementation to it.

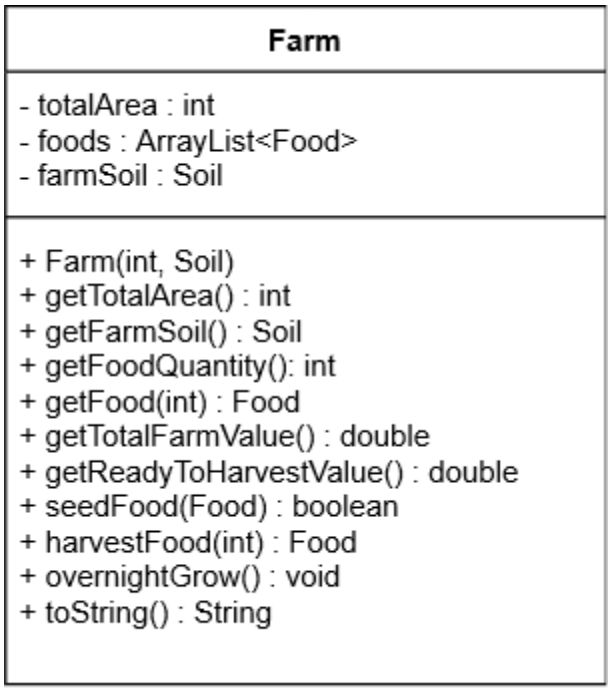
- Vegetable(…) : Vegetables should be initialized with a default preferred soil to Soil.Loam
- Fruit(…): Fruits should be initialized with a default preferred soil to Silt
- Vegetable Grow: Vegetables will grow just like regular foods if they are in their preferred soil. If they are not seeded in their preferred soil (e.g.: Clay) they should skip every other day. A vegetable in any soil that is not Loam should take double the amount of time to grow. You should not increase the daysSincePotted counter when you skip days.
- Fruit Grow: Fruits grow similar to vegetables. However, if they are seeded in a different soil than the preferred one, fruits will skip two days instead of only one.
- **Do not alter the daysToMature variable after it has been initialized in the constructor.** If you want, you can defined it as final but it is not required.

Carrot	Cabbage	Apple
+ Carrot()	+ Cabbage()	+ Apple()

- Carrots require 100 square meters, 15 days to mature, and sell for 750.32 CAD. Their preferred soil is Sand.
- Apples require 275 square meters, 15 days to mature, and sell for 1250.00 CAD. Their preferred soil is Loam.
- Cabbages (My Cabbages!) require 50 square meters, 10 days to mature, and sell for 239.75 CAD. Their preferred soil is Clay

The Farm and Apple Farm

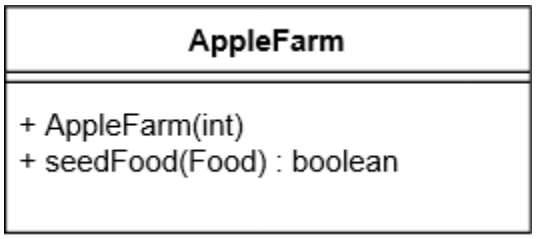
Farms can hold different types of food as long as they have enough space for each food. Each farm will have only one type of soil, and once they are created, you cannot change the soil.



Methods:

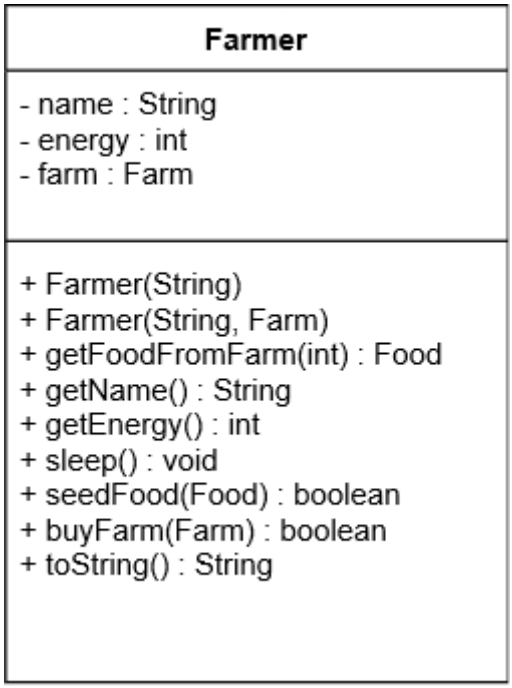
- Farm(int, Soil): Creates a new farm given its area in square meters and its soil type.
- getTotalFarmValue() : computes the total farm value combining the selling price of all the foods seeded at the farm
- getReatyToHarverstValue(): computes the farm value only considering the foods that are ready to harvest
- getFood(int): Returns the food at the given index or null if the index is invalid. **Please note you should not create a copy of the food you are returning since we have not talked about exceptions.** Return the reference for the food object stored in the ArrayList directly.
- seedFood(Food) : Tries to seed food in the farm. If the farm does not have enough area left for the new food, return false and do not add the food to the farm. If the farm has enough space, add the food to the farm and return true.
- harvestFood(int) : tries to harvest food at the int index from the farm. Foods can only be farmed if the time since potted is greater or equal than the time they need to mature. If the food is harvested, the method should remove it from the farm and return the food object. Otherwise, the method does nothing and returns null
- overnightGrow() : iterates over all foods on the farm and make them grow one cycle.

- toString() : It should print the farm info and the information of all food on the farm:
  Total farm value: 2240.07
  Ready to harvest value: 0.00
  Food available:
  0 - Cabbage (Clay) - 0/10 days
  1 - Carrot (Sand) - 0/15 days
  2 - Apple (Loam) - 0/15 days



Apple farms should have Loam as their soil. Farmers cannot seed any food that is not Apples on the farm. If a farmer tries to seed something that is not an apple (such as Cabbages!), the method should return false.

Farmer



Farmers have a name and can own a farm. Farmer objects can be initialized already owning a farm or without one. Farmer objects should always be initialized with energy to 100.

- sleep() : recovers the farmer energy by 35 (energy cannot be greater than 100) and start one growth cycle at the farm
- seedFood(Food) : farmers can seed food on the farm (assuming they own one) if they have enough energy.
    - Vegetables cost 30 energy points to seed
    - Fruits cost 50 energy points to seed

    If the farmer does not own a farm, does not have enough energy, or the farm does not have enough space, you should return false. Otherwise, return true.

- buyFarm(Farm) : buys a farm if the farmer doesn't own a farm. The farmer cannot buy a farm if they already have a farm
- toString()

Farmer: Joe Energy left: 100/100 Joe owns no farm	Farmer: Joe Energy left: 5/100 Farm info: Total farm value: 3250.32 Ready to harvest value: 0.00 Food available: 0 - Apple (Loam) - 1/15 days 1 - Apple (Loam) - 0/15 days 2 - Carrot (Sand) - 0/15 days

Grading Scheme

Each problem on the assignment will be graded based on three criteria:

Functionality

"Does it work according to specifications?" This is determined in an automated fashion by running your program on a number of inputs and ensuring that the outputs match the expected outputs. The score is determined based on the number of tests that your program passes.

Quality of Solution

"Is it a good solution?" This considers whether the solution is correct, efficient, covers boundary conditions, does not have any obvious bugs, etc. This is determined by visual inspection of the code. Initially full marks are given to each solution and marks are deducted based on faults found in the solution.

Code Clarity

"Is it well written?" This considers whether the solution is properly formatted, well-documented, and follows coding style guidelines (https://web.cs.dal.ca/~franz/CodingStyle.html).

If your program does not compile, it is considered non-functional and of extremely poor quality, meaning you will receive 0 for the solution.

PROBLEM	POINTS
FUNCTIONALITY	79
QUALITY OF SOLUTION & CODE CLARITY	21
TOTAL	**100**