**COMP 1112 Object Oriented Programing**
**Spring 2022, Project#2**
**Due: 1 June, 2022 23:59**

IŞIK UNIVERSITY
COMPUTER SCIENCE AND ENGINEERING

**DO NOT WAIT FOR THE LAST TWO DAYS!!**

**NEVER EVER SHARE YOUR DESIGN and CODE WITH ANYONE ELSE. ANY KIND OF PLAGIARISM IS NOT ALLOWED and WILL NOT BE TOLERATED.**

**THIS PROJECT IS A GROUP PROJECT. PROJECT GROUPS ARE ANNOUCED AT THE BLACKBOARD.**

## Agricultural Management System

**General explanation:** You are supposed to **design** and **implement** an agricultural crop management system. There exist three entities in this system**: Crop, Store,** and **Supplier**. Crops have two types: **Fruit** and **Vegetable**. Only fruits are stored (kept in coolers and listed) in the stores or in the suppliers after cultivation, and suppliers can buy the existing fruits available in the store or sell their own fruits and then those sold fruits are kept in the store. There can be a list of crops, stores and suppliers. When the program starts, the list of crops, stores and suppliers are read from relevant text files. The content of these files are given at the end of this word file.

The first entity in this system is **Crop** with following attributes and functionalities:

1) Abstract **Crop** class:

**Attributes:**

- **name**
- **weight** (of the whole crop in Kilograms)
- **cultivatedSeason**: the name of season in which the crop has been cultivated.

**Note:** The attributes mentioned above are initialized at construction time and cannot be changed later.
**Methods:**

- **Constructor**(s)
- **Abstract toString**() as an abstract method which must be implemented in subclasses of the Crop class.
- **Abstract consumeIt()**: "vegetables are cooked" and "fruits are consumed raw".
- **Abstract storeIt**(): stores the whole crop in the store (put into the related list) by invoking the relevant method from the store class; can be used only for fruit subclass, and for vegetable subclass it throws a **CanNotBeStoredException.**

Basically, there are two subclasses of crops: **Fruit** and **Vegetable**. Only Fruits can be stored in the stores (listed), bought or sold by suppliers (explained below).

- **Fruit class** (inherits from Crop and implements Comparable interface):
    - Attributes: **taste, price (tl/kg).**
    - An object attribute **CropKeeper**, that is implemented as an **interface,** can be either **Store** or **Supplier.**
- **Vegetable class** (inherits from Crop and implements Comparable interface):
    - Attributes: **cultivatedRegion** indicates the name of the city cultivating this crop such as Adana, Trabzon, ... (initialized in constructors and not changed later).

"The bearing of a child takes nine months, no matter how many women are assigned."
*Fred Brooks*

- An object attribute **CropKeeper**, that is implemented as an **interface,** can be either **Store** or **Supplier.**

**Note:** Comparable interface should be implemented by Fruit and Vegetable classes. Two fruits are the same if their name and color are equal. Two vegetables are the same only if their name are equal. If two fruits or vegetables are equal, **compareTo** method returns 0, otherwise it returns the weight difference between two crops (current crop weight – input crop weight).

2) **Store** class:

It implements **CropKeeper** that has a **howToStore** method. A Store stores "fruits in large refrigerated cooler rooms" and it keeps "vegetables in sheds, not listed".

Attributes:

- **ID** (must be unique, not changeable and always starts with digit 5)
- **name**
- **maxCapacityArea** (square meter)
- **usedCapacityArea** (square meter). Will be updated when a new fruit instance is added to or removed from the store.
- **KGperSquareMeter**: indicates how many kilograms of any fruit instance can be stored in one square meter of this store (Default value: 10).
- **fruitList:** List of fruit instances stored in this store as an array list of fruits. Only different types of fruits can be stored in the same store. For example, red apple cannot be stored as a different object in store if another red apple object already exists in it, instead the amount of the new red apple is added to the existing one. But green apple and red apple can both be stored in this store as two different object.

Methods:

- **Constructor**(s)
- **Setters/Getters**
- **availabeCapacity**(): Returns the free capacity in terms of square meters.
- **canBeStored**(Fruit f): Receives a fruit object and returns a Boolean value indicating if that it can be stored in the store or not. A new fruit object cannot be added to store if there is not enough capacity in it.
- **importCrop**(Fruit f): Adds the whole amount of input fruit object to the store if there is enough free space available in the store; otherwise, it throws an **CapacityNotEnoughException** and returns.
- **exportCrop**(Fruit f): Removes the whole amount of Fruit f from the FruitList (if it already exists in it) and frees its occupied space in the store; if it does not exist in store, throws an **FruitNotFoundException** and returns.

Suppliers remove (export) available fruits from the store or add (import) them to the store. Note that either the whole amount or none of a fruit is bought by a Supplier. **No partial export or import is possible.** Also a specific fruit can be bought by a Supplier if he does not have it in his cropList. When selling a fruit, the fruit should be removed from this Supplier's cropList and the earned money should be added to the Supplier's Budget. Also, the sold crop should be added to the Store only if it is a fruit. Vegetables will not be stores after they are sold. On the other hand, when buying fruit from store, the relevant fields of the Supplier should be updated, e.g., budget and cropList.

3) **<u>Supplier</u>** class:

It implements **CropKeeper** that has a **howToStore** method. A Supplier keeps "fruits in big refrigerators" and it keeps "vegetables in the field booths".

Attribute list:

- **name**
- **ID (must be unique and always starts with digit 1)**
- **budget**: the amount of money in liras that the Supplier has in his account.
- **cropList**: list of crops that the Supplier owns.

Methods:

- **Constructor**(s)
- **Setters/Getters**
- **buyCrop**(Crop c): if the input crop is available in the store, and the Supplier has enough money to buy it, then the Supplier buys it by calling related methods and doing related field updates; otherwise, **SupplierHasNotEnougMoneyException** or **FruitNotAvailableException** accordingly. Absolutely the bought fruit should be added to the Supplier's **fruitList.**
- **sellCrop**(Crop c): if the input crop exists in the **fruitList** of this Supplier, sell it and store it in one of the available stores and update the relevant fields; otherwise, the method throws an **FruitNotFoundException** and returns.

**Note**: CropKeeper interface should be implemented by Store and Supplier.

4) `TestClass`: used as the main interface. When the program starts running in main method, the list of Suppliers and Crops are loaded from relevant files (given to your program as input). Then, your code should work correctly in a flow of menus suggested below.
Main menu:
Please make a choice(0 to 91)

- (Press 1) **Display all suppliers** and their crop list, how they are stored and how they are consumed.
- (Press 2) **Display all stores** and their fruit list, how they are stored and how they are consumed.
- (Press 3) **Buy a fruit crop** for a Supplier and add it to the a stores's fruitList**,** removing it from the related store. Ask the value of required fields from the user.
- (Press 4) **Sell a fruit crop** of a Supplier and remove it from the supplier's cropList**,** adding it to the related store's fruitList. Ask the value of required fields from the user.
- (Press 5) **Remove a fruit from a store**. Ask the required fields from the user.
- (Press 6) **Remove a crop from a supplier**. Ask the required fields from the user.
- (Press 7) **Add crop**. Ask the required fields from the user and add the new crop to a store or a supplier.
- (Press 8) **Show remaining budget**. Print remaining budget of a given supplier.

- (Press 9) **Show remaining capacity**. Print remaining capacity of a given store.
- (Press 0) **Quit** : quits the app.

The menu shown above should continue running until the user presses 0.

**Note**: If you believe any attribute is missing in any class designed above, or new arguments are necessary to be added to some methods, feel free to add them and make any further assumptions.

The list of **Suppliers.txt**: [Supplier name, Supplier ID, budget]
ArazMeyve, 1133, 1000
AylarTarim, 1298, 1500
HasanBey, 1322, 200
ZehraCiftci, 1429, 1250

The list of **Stores.txt**: [Store name, Store ID, maxCapacityArea(m2), KGperSquareMeter]
Migros, 5343, 1000, 12
File, 5967, 1200, 10

Hint on **Crops.txt**:

Name, type, kilogram, season, taste, price, ID of cropKeeper (for fruits)
Name, type, kilogram, CityName, ID of cropKeeper (for vegetables)

The initial content of **Crops.txt**:

RedApple, fruit, 45, winter, sweet, 3, 1133
Orange, fruit, 50, autumn, sour, 4, 5967
Kiwi, fruit, 10, autumn, sour, 10, 1133
Parsley , vegetable, 25, Samsun, 1429
Mint, vegetable, 15, Adana, 1429
GreenApple, fruit, 25, winter, sweet, 6, 1133
Orange, fruit, 20, autumn, sour, 4, 1322
GreenApple, fruit, 5, winter, sweet, 6, 5343
GreenBeans, vegetable, 22, Bursa, 1322
Banana, fruit, 63, summer, sweet, 12, 5343

**Hint:** When you read the crops from the file and create related crop objects, you should also add them to the related CropKeeper's list.

Good luck.

"The bearing of a child takes nine months, no matter how many women are assigned."
*Fred Brooks*