Problem 2. Farmer's Creed

Ubisoft's new Assassin's Creed will be out soon and we need an answer of our own! After a long meeting this is the game we cooked up: there is a **farm** and in that farm there are **animals** and **plants**. The farm can **perform different actions** on the animals and plants it holds (**watering** plants, **feeding** animals, etc.). In time, the animals and plants may **produce** things such as **milk** (from cows) or **tobacco** (from tobacco plants).

You are given an API which simulates life on the farm. The API contains the following components:

- **FarmSimulator** – contains the **farm** used for the simulation. **Reads** user input commands and **consults** to the **farm** whenever something needs to be done
- **Interfaces**:
  - **IFarm** – defines the **functionality** a farm should support
  - **IProduct –** defines a **product**
  - **IEdible –** defines an **edible** (eatable) product
  - **IProductProduceable** – defines functionality to produce a product
- **Game Objects**:
  - **`GameObject`** – base class for all game objects, holds **`Id`**
  - **`FarmUnit`** – base class for all farm units. Holds **`Health`** and **`IsAlive`**. Implements **`IProductProduceable`** interface
  - **`Product`** – represents a product which **IProductProduceable** units **produce**. Implements **IProduct** interface
    - **`FoodProduct`** – product which can be **eaten** by animals. Implements **IEdible** interface

Currently, the **FarmSimulator** supports 3 commands:

- **`create (farmId)`** – creates the farm the game will use
- **`add (unitType) (unitId)`** – adds a given unit to the farm
  - **`add Grain (id)`** – adds a **food product** of **organic** type with **quantity 10** and **HealthEffect 2** to the farm
- **`status`** – prints information about an object on the console
  - **`status animal (animalId)`** – prints information about a specific animal
  - **`status plant (plantId)`** – prints information about a specific plant
  - **`status product (productId)`** – prints information about a specific product
  - **`status farm`** – prints information about the farm

Your task is to **extend** the given API so that it supports a **real farm simulation**.

Design the Class Hierarchy

The following classes should be implemented:

- Implement **`Plant`**:
  - Holds **`int GrowTime`**, **`void Water()`**, **`void Wither()`**, **`void Grow()`**
  - A plant is a farm unit which can **grow** by **decreasing its `GrowTime` by 1**. **`GrowTime`** represents the time it takes for a plant to grow.

    - Each plant produces a specific **product** when **exploited**
    - Plants increase their **Health** when **watered** by **2** and decrease it when they **wither** by **1**
- Implement **`TobaccoPlant`**:
    - Plant with base **`Health`** of **`12`** and **`GrowTime`** of **`4`**
    - Grows **twice** as fast as other plants
    - Produces a product with **`ProductType`** of **`Tobacco`** and **`Quantity`** of **`10`**
    - Cannot produce while growing or if dead
- Implement **`FoodPlant`**:
    - Base class for plants which produce **food products**
    - Increases its **`Health`** by **`1`** when watered
    - Withers **twice** as fast as other plants.
- Implement **`CherryTree`**:
    - Food plant with base **`Health`** of **`14`** and **`GrowTime`** of **`3`**
    - Produces a food product with **`ProductType`** of **`Cherry`**, **`FoodType`** of **`Organic`**, **`Quantity`** of **`4`** and **`HealthEffect`** of **`2`**
    - Cannot produce if dead
- Implement **`Animal`**:
    - Holds **`void Eat(IEdible edibleProduct, int quantity)`**
    - Eats **`quantity`** amount of the **`edibleProduct`**
    - Holds **`void Starve()`**
    - Loses **`1 Health`** each time it **starves**
- Implement **`Cow`**:
    - Animal with base **`Health`** of **`15`**
    - Produces a product with **`ProductType`** of **`Milk`**, **`FoodType`** of **`Organic`**, **`HealthEffect`** of **`4`** and **`Quantity`** of **`6`**
    - Loses **`2 Health`** when producing **milk**
    - If the cow eats **Organic** food, it **gains** the food's **`HealthEffect * productQuantity`**. If the food is **Meat**, it **loses** the **`HealthEffect * productQuantity`** from its own **`Health`**
    - Cannot produce if dead
- Implement **`Swine`**:
    - Animal with base **`Health`** of **`20`**
    - Produces a product with **`ProductType`** of **`PorkMeat`**, **`FoodType`** of **`Meat`**, **`HealthEffect`** of **`5`** and **`ProductionQuantity`** of **`1`**
    - **Dies** after producing a product
    - Starves **3 times** as fast as other animals
    - Eats both **Organic** and **Meat** food and gains **twice** the **`HealthEffect`**
    - Cannot produce if dead
- Implement **`Farm`:**
    - Implements the **`IFarm`** interface
    - Performs all operations (**feeding**, **exploiting**, **watering**, etc.) on the units it holds (**animals**, **plants** and **products**)
    - All plants which are added to the farm start **growing**
    - All live plants which have **already grown** start to **wither**
    - All live animals start **starving**
    - If a product with a **given `Id`** is **added** to the farm and another product with the **same `Id`** already exists, only the **quantity** of the existing product is increased

Commands

The following commands should be implemented:

- **add** commands for the following units:
  - **`add CherryTree (id)`** – adds a new **`CherryPlant`** to the farm
  - **`add TobaccoPlant (id)`** – adds a new **`TobaccoPlant`** to the farm
  - **`add Cow (id)`** – adds a new **`Cow`** to the farm
  - **`add Swine (id)`** – adds a new **`Swine`** to the farm
- **`feed (animalId) (foodId) (quantity)`** – feeds animal **`animalId`** with **`quantity`** of food **`foodId`** and reduces the food's quantity
- **`water (plantId)`** – waters plant **`plantId`**
- **`exploit animal/plant (animalId/plantId)`** – **exploits** (gets the product from) an **animal**/**plant** and adds the product to the farm

The **`status`** commands should print information about a unit in the following formats:

- **Animals** (either blue or red text can be printed at a time):

```
--(ClassType) (Id), Health: (Health) / DEAD
```

- **Plants**:

```
--(ClassType) (Id), Health: (Health), Grown: (Yes/No) / DEAD
```

- **Products**:

```
--(ClassType) (Id), Quantity: (Quantity), Product Type: (ProductType),
Food Type: (FoodType), Health Effect: (HealthEffect)
```

- **Farm**:

```
--(ClassType) (Id)
--(information about animals)
--(information about plants)
--(information about products)
```

  The farm should print each of its units on a **separate line**. Print only animals which are **alive**, sorted by **`Id`**. Print only plants which are **alive**, sorted by **`Health`** and then by **`Id`**. Print all products, sorted by their **`ProductType`** (alphabetically), then by **`Quantity`** in **descending order** and finally by **`Id`**.

Extend the API by implementing the necessary functionality using the best practices of OOP. Avoid code duplication through inheritance and abstraction. You are **NOT** allowed to directly edit the **FarmSimulator** and the **provided interfaces**. You may edit the Main() method and the unimplemented classes.

Additional Notes

- Products produced by animals/plants should have an **Id** equal to the producing unit's **Id + "Product"**. E.g. Cow ***Krava*** produces Food with Id ***KravaProduct***.
- Live animals **starve** and live plants **grow** or **wither** each time a command is processed, **except for status commands**. (See **`FarmSimulator`** for details on how it works). **Edit: Use the UpdateFarmState() method for this purpose.**
- **Exceptions** should be **thrown** whenever an invalid operation is executed (e.g. milking a dead cow, using a product with negative quantity, exploiting plants or animals when it is not permitted, etc.).

Input	Output
create Tarapontii-Farm add Grain Jito add CherryTree Chereshata add Cow Krava add Cow Govedo add Swine Prasio add TobaccoPlant Tutun exploit animal Govedo feed Prasio GovedoProduct 3 status animal Prasio exploit plant Tutun status product GovedoProduct status plant Chereshata exploit plant Chereshata exploit plant Chereshata status plant Chereshata status product ChereshataProduct status animal Prasio water Tutun status plant Tutun exploit animal Prasio status animal Prasio status animal Krava feed Krava PrasioProduct 1 status animal Krava status farm END	--Swine Prasio, Health: 32 --Food GovedoProduct, Quantity: 3, Product Type: Milk, Food Type: Organic, Health Effect: 4 --CherryTree Chereshata, Health: 4, Grown: Yes --CherryTree Chereshata, DEAD --Food ChereshataProduct, Quantity: 8, Product Type: Cherry, Food Type: Organic, Health Effect: 2 --Swine Prasio, Health: 23 --TobaccoPlant Tutun, Health: 9, Grown: Yes --Swine Prasio, DEAD --Cow Krava, Health: 4 --Cow Krava, DEAD --Farm Tarapontii-Farm --Cow Govedo, Health: 2 --TobaccoPlant Tutun, Health: 7, Grown: Yes --Food ChereshataProduct, Quantity: 8, Product Type: Cherry, Food Type: Organic, Health Effect: 2 --Food Jito, Quantity: 10, Product Type: Grain, Food Type: Organic, Health Effect: 2 --Food GovedoProduct, Quantity: 3, Product Type: Milk, Food Type: Organic, Health Effect: 4 --Food PrasioProduct, Quantity: 0, Product Type: PorkMeat, Food Type: Meat, Health Effect: 5 --Product TutunProduct, Quantity: 10, Product Type: Tobacco