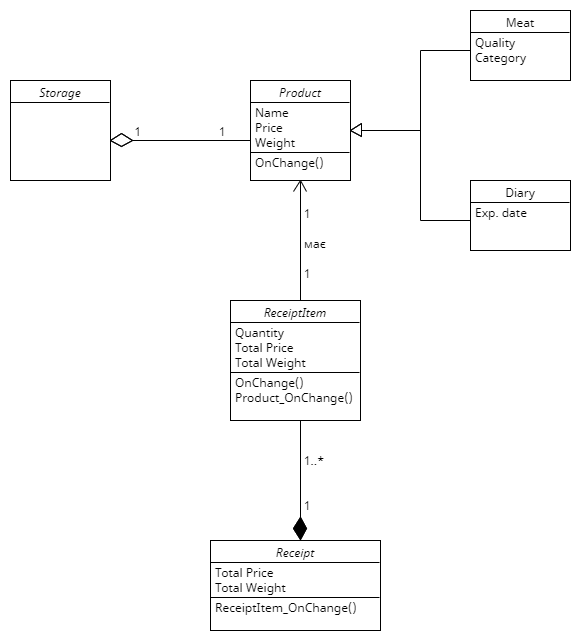
До початку проектування діаграма класів бібліотеки продуктів виглядала ось так:

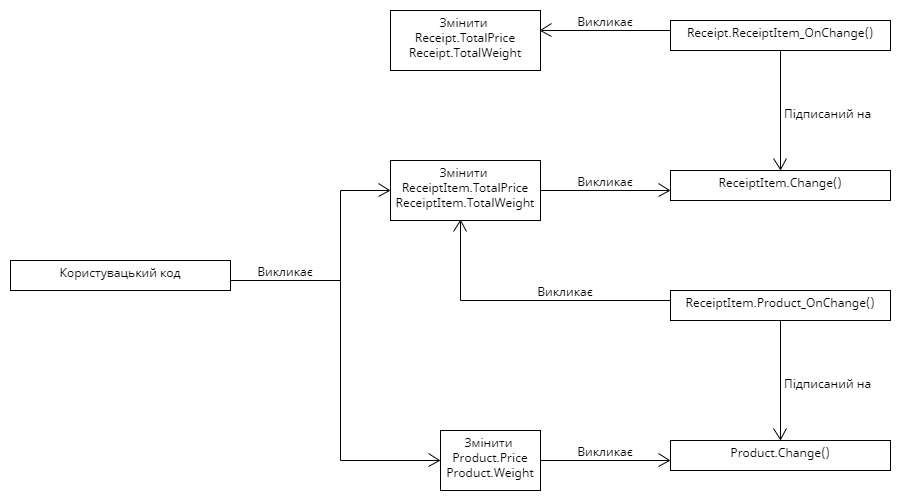


Product – абстрактний клас продукту

ReceiptItem – стрічка у чеку, замовленні із кількістю продукту

Receipt – чек зі стрічками продуктів

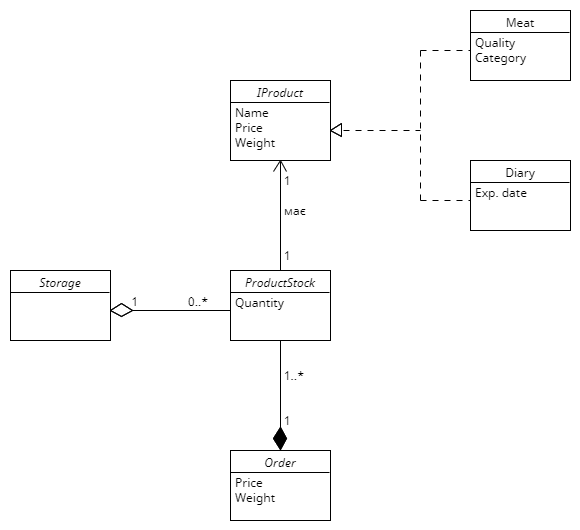
Через всю архітектуру проходить ідея ієрархії делегатів між класами Product -> ReceiptItem -> Receipt. Тобто є випадок, в якому класи вже заповненні (маємо певний чек), і якщо зміниться вага або ціна якогось елементу, то це змінить всі елементи зверху ієрархії.



Проблеми цього механізму, на мою думку є:

1. Виникає потреба у абстрактному класі Product для неповторюваності коду цього механізму.
2. Потрібно піклуватися, щоб обьекти Product існували в одному екземплярі на всю програму, щоб це працювало
3. Клас Storage зберігає обьекти Product при тому, що цей механізм ніяк не відноситься до нього
4. Сумнівність взагалі потреби у такому механізмі

Нова архітектура:



IProduct – Ідея товару, який повинен мати назву, ціну та вагу

ProductStock – Сутність яка має товар та його кількість

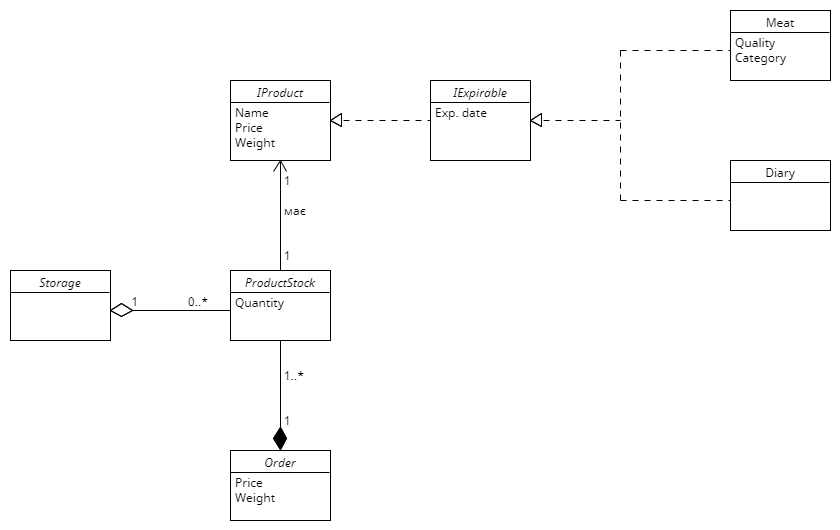
Order – Замовлення (може також виконувати роль чеку)

Storage – Фізичний склад продуктів

Order та Storage є узагальненими (generic) через посередника ProductStock.

Погляд у майбутнє

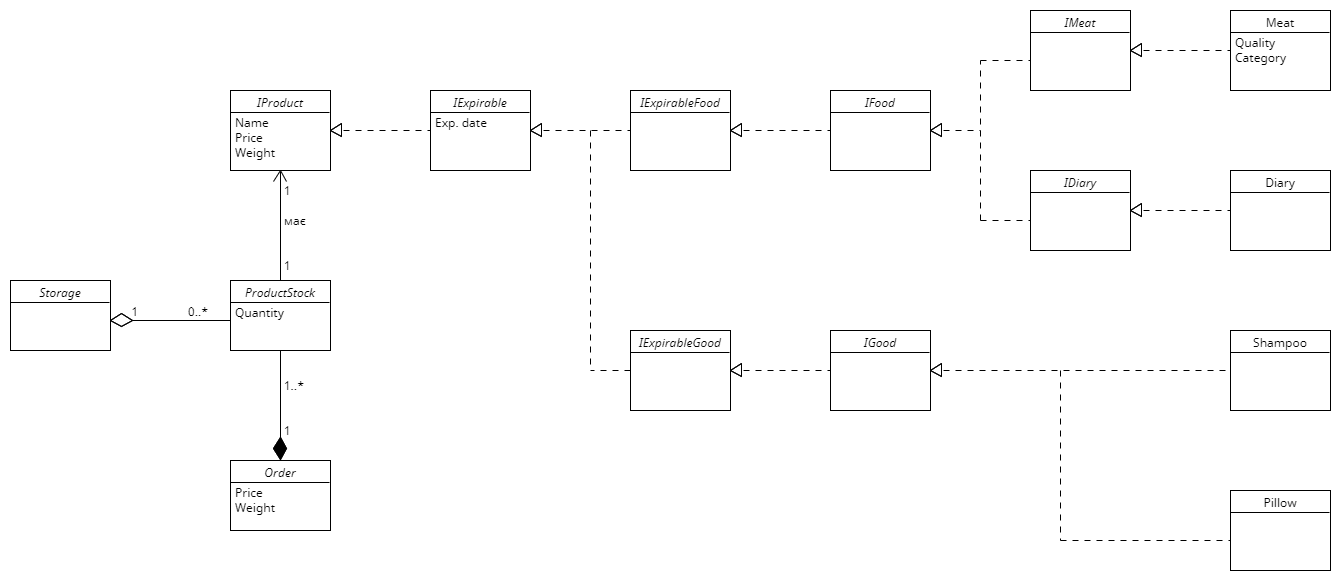
Якщо ми захотіли, щоб м'ясо також мало термін придатності, ми можемо відокремити інтерфейс IExpirable



Наш магазин почав продавати шампуні та подушки.

І те і інше має термін придатності, але він не має такого значення як, наприклад, у молочних продуктів, тому можна відокремити інтерфейси IFood та IGood та додаткові (допоміжні) IExpirableFood та IExpirableGood для читабельності коду.

Це забезпечить, що у колекцію типу IExpirableFood не зможе «заблукати» обьект, який імплементує інтерфейс, який наслідується від IExpirableGood.



Зображення додано у папку.

Потім ми віришили почати продавати вогнегасники.

Вогнегасник відноситься до IGood але має термін придатності, який має більше значення, порівняно з шампунем та подушками.

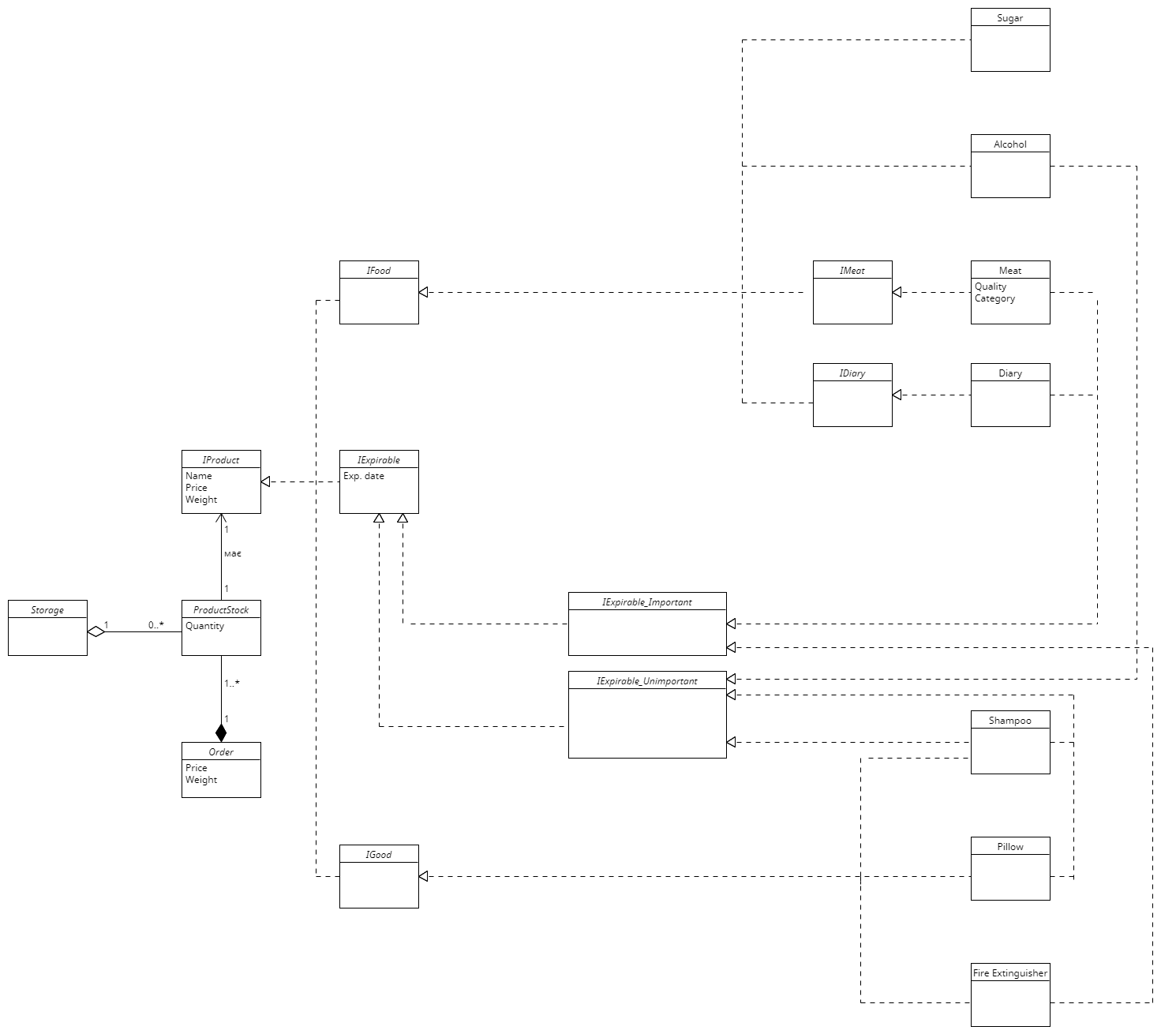
Потім магазин почав продавати алкоголь, який є IFood, але його термін придатності не таким важливим як, наприклад, у м’яса.

Потім магазин почав продавати цукор, який є IFood, але терміну придатності не має взагалі.

Виникає потреба у відділенні додаткової незалежної ієрархії, яка стосується терміну придатності.

Створено IExpirable\_Important та IExpirable\_Unimportant.

Тепер класи можуть на свій роздум незалежно відносити себе до «їжі» або «не-їжі» та «з важливим терміном придатності» або «з неважливим терміном придатності».



Зображення додано у папку.

Після всіх цих ускладнень, поточний функціонал все ще працює, завдяки базового інтерфейсу IProduct. Завдяки ньому архітектуру продуктів можна і далі **розширяти** (але не модифікувати!) без виникнення багів у старому коді.

Це слідує із другого принципу SOLID-у – “Open-Closed Principle”.