**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**КАФЕДРА САПР**

**ОТЧЕТ**

**по лабораторной работе №9**

**по дисциплине «Объектно-ориентированное программирование»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9301 |  | Примакова Е.Е.. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2021

# Цель работы

Получить навыки практической работы с классами в языке C#. Закрепить изученный за семестр теоретический материал по теме. Самостоятельно разработать систему классов.

# Анализ задачи

Необходимо реализовать систему классов для представления склада продуктов

# Теоритическая часть

Абстрактные классы — классы, реализованные с ключевым словом abstract. Ключевая особенность этих классов состоит в том, что создавать экземпляры этих классов нельзя. Данные классы используются для описания общих свойств некоторых классов объектов, после чего другие классы их наследуют. Таким образом, у классов наследуются поля и методы, а при необходимости что-то поменять меняется только абстрактный класс, а не все классы, наследовавшие его, по отдельности.

Наследование классов. Благодаря наследованию один класс может унаследовать функциональность другого класса. Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений: не поддерживается множественное наследование — класс может наследоваться только от одного класса; при создании производного класса надо учитывать тип доступа к базовому классу — тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public, однако следует также учитывать, что если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производный класс может наследовать только от класса, который имеет модификатор public; если класс объявлен с модификатором sealed, то от этого класса нельзя наследовать и создавать производные классы; нельзя наследовать от статического класса.

Интерфейс. Для определения интерфейса используется ключевое слово interface. Интерфейс представляет ссылочный тип, который может определять некоторый функционал - набор методов и свойств без реализации. Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы. Методы и свойства интерфейса могут не иметь реализации, в этом они сближаются с абстрактными методами и свойствами абстрактных классов. По умолчанию спецификатор доступа public.

# Формальная постановка задачи

## Вариант 17:

Разработать программу для представления склада продуктов

Упражнение 1 — Разработка структур данных для заданной предметной области

В этом упражнении необходимо для заданной предметной области (первый раздел курсовой работы) разработать несколько классов и интерфейсов. Желательно использовать абстрактный класс. Применить механизмы наследования. Результаты представить с помощью диаграммы классов и спецификации.

Упражнение 2 – Разработка программы на основе созданных в упр.1 структур данных

В этом упражнении необходимо для заданной предметной области разработать программу. Необходимо использовать механизмы наследования с применением интерфейсов и абстрактных классов.

# Текст программы

Упражнение 1:

**using** System;

**using** System.Collections.Generic;

**namespace** lab9

{

**enum** ContainerType // тип хранения

{

COLD, // холодильник

WET, // сухой склад

DRY, // влажный склад

}

**enum** Unit // единица измерения

{

piece, // штуки

kg, // килограммы

gram, // граммы

pack, // упаковки

liter, // литры

ml, // милилитры

}

**abstract** **class** Piece // единица

{

**public** Unit unit; // единица измерения -- в них количество товара и за 1 единицу этого стоимость

**public** **float** amount; // количество тоара в единицах unit -- добавить ограничения на ><0

}

**class** Item : Piece // наименование

{

**public** **double** price; // цена за единицу товара

**public** **uint** due\_date; // [1;366] срок годности

**public** String article; // артикул товара (строка потому что артикул может содержать буквы и знаки препинания)

**public** ContainerType container; // место хранения -- сухой/влажный склад или холодильник

**public** Item(**double** price = 0.0, **uint** due\_date = 0, String article = "", ContainerType container = 0, Unit unit = 0, **float** amount = 0)

{

**this**.price = price; // инициализация поля цена

**this**.due\_date = due\_date; // инициализация поля срок годности

**this**.article = article; // инициализация поля артикул

**this**.container = container; // инициализация поля место хранения

**this**.unit = unit; // инициализация поля единица измерения

**this**.amount = amount; // инициализация поля количество

}

**public** **float** How\_Much() // возвращает значение поля количество

{

**return** amount;

}

**public** **bool** Remove(**float** amount) // уменьшает количество на заданное значение, если это возможно

{

**if** (**this**.amount < amount)

{

**return** **false**;

}

**this**.amount -= amount;

**return** **true**;

}

**public** **void** Add(**float** amount) // увеличивает количество на заданное значение

{

**this**.amount += amount;

}

}

**class** Storage // склад

{

**public** List<Item> cold\_storage; // список товаров, хранящихся в холодильнике

**public** List<Item> wet\_storage; // список товаров на влажном складе

**public** List<Item> dry\_storage; // список товаров в сухом складе

**public** Storage()

{

cold\_storage = **new** List<Item>();

wet\_storage = **new** List<Item>();

dry\_storage = **new** List<Item>();

}

**public** **bool** Find(String article, ContainerType container, **ref** Item item) // поиск элемента на складе типа container

{

**switch** (container) // в зависимости от типа контейнера

{

// просматриваем каждый элемент списка хранилища, пока не найдём тот элемент, который нам нужен

**case** ContainerType.COLD:

{

**foreach** (Item a **in** cold\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**break**;

}

**case** ContainerType.WET:

{

**foreach** (Item a **in** wet\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**break**;

}

**case** ContainerType.DRY:

{

**foreach** (Item a **in** dry\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**break**;

}

}

**return** **false**; // если элемент не был найден

}

**public** **bool** Find(String article, **ref** Item item) // поиск элемента на складе всех типов

{

// проходим по всем хранилищам, пока не найдём элемент

**foreach** (Item a **in** cold\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**foreach** (Item a **in** wet\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**foreach** (Item a **in** dry\_storage)

{

**if** (a.article == article)

{

item = a;

**return** **true**;

}

}

**return** **false**; // если элемент не был найден

}

**public** **bool** Buy(String article, ContainerType container, **float** amount)

{

Item item = **new** Item(); // переменная для хранения наименования склада

**if**(Find(article, container, **ref** item)) // ищем элемент

{

**return** item.Remove(amount); // если вохможно, убираем нужное количество, если нет -- вернём false

}

**return** **false**; // если элемент не был найден

}

**public** **bool** Sell(String article, ContainerType container, **float** amount)

{

Item item = **new** Item();

**if** (Find(article, container, **ref** item))

{

item.Add(amount); // добавляем элементы

**return** **true**;// если элемент был найден

}

**return** **false**; // если элемент не был найден

}

**public** **void** Add\_Item(**double** price = 0.0, **uint** due\_date = 0, String article = "",

ContainerType container = 0, Unit unit = 0, **float** amount = 0) // добавление элемента на склад

{

// создаём элемент и добавляем его на свой склад

Item item = **new** Item(price, due\_date, article, container, unit, amount);

**switch** (container)

{

**case** ContainerType.COLD:

{

cold\_storage.Add(item);

**break**;

}

**case** ContainerType.WET:

{

wet\_storage.Add(item);

**break**;

}

**case** ContainerType.DRY:

{

dry\_storage.Add(item);

**break**;

}

}

}

**public** **void** Add\_Item(Item item) // добавление элемента на склад

{

// добавляем элемент на свой склад

**switch** (item.container)

{

**case** ContainerType.COLD:

{

cold\_storage.Add(item);

**return**;

}

**case** ContainerType.DRY:

{

dry\_storage.Add(item);

**return**;

}

**case** ContainerType.WET:

{

wet\_storage.Add(item);

**return**;

}

}

}

**public** **bool** Delete\_Item(String article, ContainerType container) // удаление наименования из списка хранения

{

Item item = **new** Item();

**if**(Find(article, container, **ref** item)) // если элемент найден

{ // удаляем его из своего контейнера

**switch** (container)

{

**case** ContainerType.COLD:

{

cold\_storage.Remove(item);

**return** **true**;

}

**case** ContainerType.WET:

{

wet\_storage.Remove(item);

**return** **true**;

}

**case** ContainerType.DRY:

{

dry\_storage.Remove(item);

**return** **true**;

}

}

}

**return** **false**; // если элемент не был найден

}

}

}

Упражнение 2:

Функция main:

**class** lab9

{

**static** **void** Main(**string**[] args)

{

Storage myStorage = **new** Storage();

myStorage.Add\_Item(10.0, 256, "мяу", ContainerType.DRY, Unit.piece, 100); // Добавляем на сухой склад 100 котиков по цене 10

myStorage.Add\_Item(100.0, 304, "гав", ContainerType.WET, Unit.pack, 100); // Добавляем на влажный склад 100 собачек по цене 100

**if**(myStorage.Buy("гав", ContainerType.WET, 1))

{

Console.WriteLine("Вы успешно купили 1 собачку!");

}

}

}

# Контрольный пример

В функции main создаётся объект класса Storage — склад, добавляются 2 наименования с помощью метода Add\_Item, производится покупка. На рисунке 1 представлен результат выполнения программы.

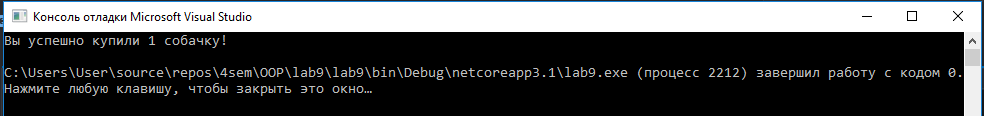


Рисунок . Результат работы функции main.

# Полученные результаты

В ходе работы была реализована структура классов, описывающая склад для хранения. Были реализованы методы добавления и удаления элементов со склада, покупки и продажи существующих. На рисунке 2 представлена диаграмма классов структуры.

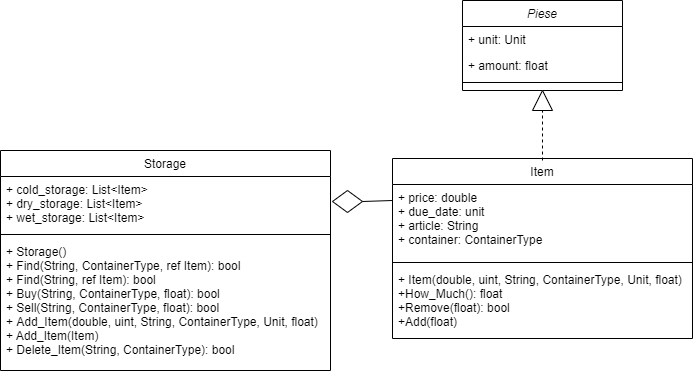


Рисунок 2. UML-диаграмма реализованных классов.

# Выводы

В ходе выполнения данной лабораторной работы были изучены принципы наследования и закреплены навыки работы с классами в C#, на их основе была реализована модель, описывающая склад продуктов.