МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем



**Лабораторна робота №2**

з дисципліни “Об’єктно-орієнтовне програмування”

***тема:*** «Реалізація основних принципів ООП мовою C#»

|  |  |
| --- | --- |
|  | Виконав Вернік Данило  студент II курсу  групи КП-22 |

Київ 2023

Мета роботи:

Ознайомитися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові С#, створенням класів, об’єктів, механізмами інкапсуляції, наслідування та поліморфізму. Вивчити механізм управління ресурсами, реалізований у .Net.

Постановка задачі

Побудувати ієрархію класів, що відтворюватимуть відношення наслідування між об’єктами реального світу (кількість класів >= 5). При цьому:

1. Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості.
3. Створити для розроблюваних класів такі конструктори:
   * конструктор за замовчанням;
   * конструктор з параметрами;
   * приватний конструктор;
   * статичний конструктор.

Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.

1. Використати віртуальні та перевизначені методи.
2. Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:
   1. реалізувати інтерфейс IDisposable;
   2. створити деструктори;
   3. забезпечити уникнення конфліктів між Dispose та деструктором.
3. Забезпечити виклики методів GC таким чином, щоб можна було простежити життєвий цикл об’єктів, що обробляються (зокрема, використати методи Collect, SupressFinalize, ReRegisterForFinalize, GetTotalMemory, GetGeneration, WaitForPendingFinalizers). Створити ситуацію, яка спровокує примусове збирання сміття GC.

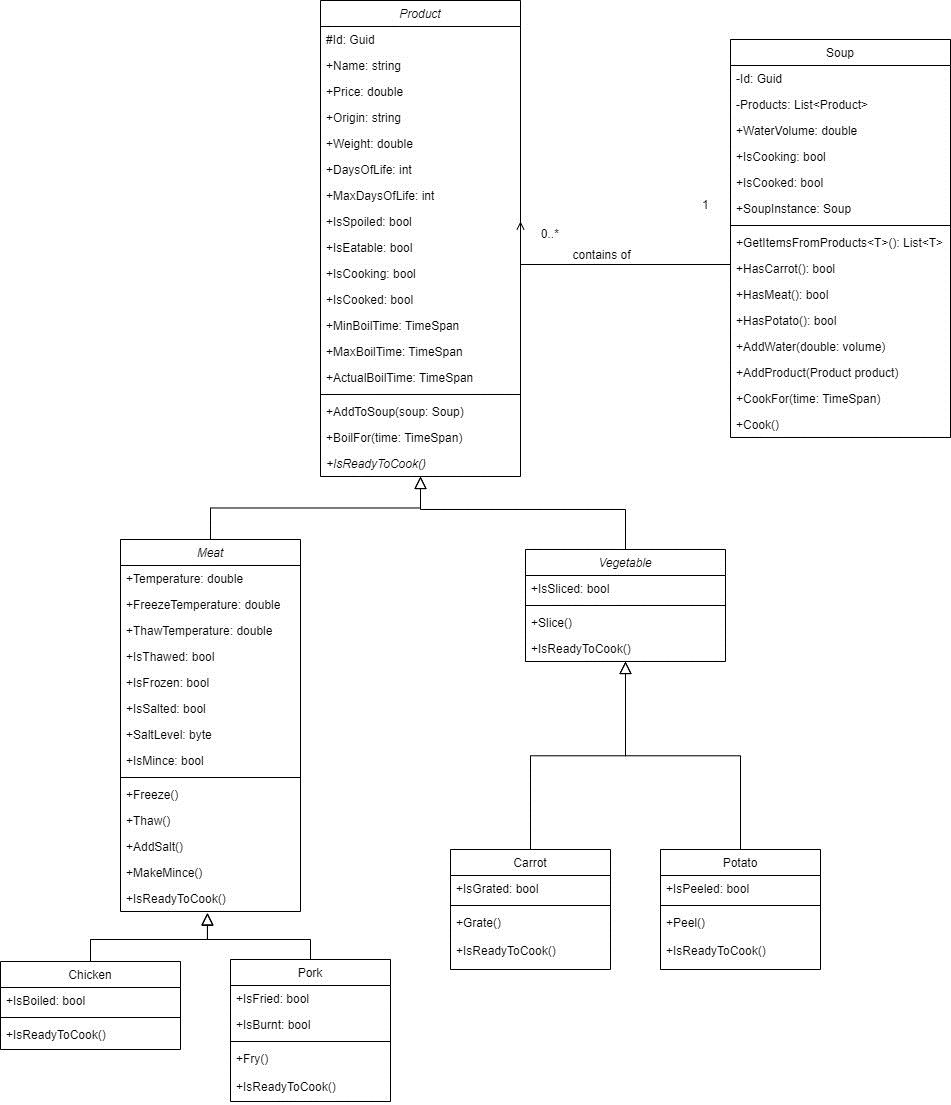
Протокол має містити: титульний аркуш, постановку задачі, UML діаграму класів, фрагменти коду, які демонструють виконання поставлених задач, висновки.

**Всього –** **10 балів**

**Хід роботи:**

Почав виконання лабораторної роботи з обрання сфери з реального світу для створення ієрархії класів – «Продукти та суп».

Розробив діаграми класів у сервісі app.diagrams.net для даної області:



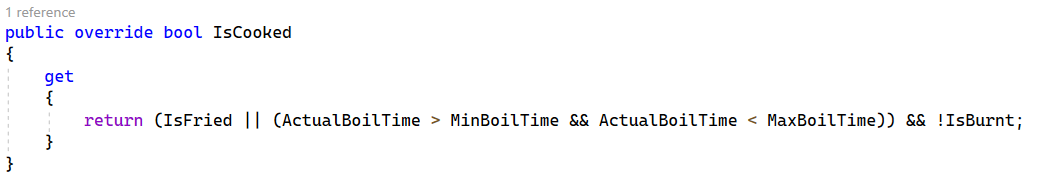
1) Звідси можна побачити, що у полів є різні модифікатори доступу, оформлені за правилами UML-diagram.

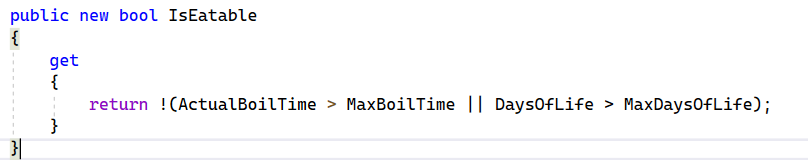
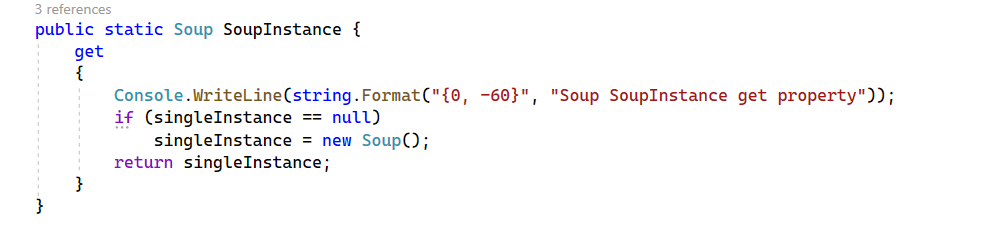
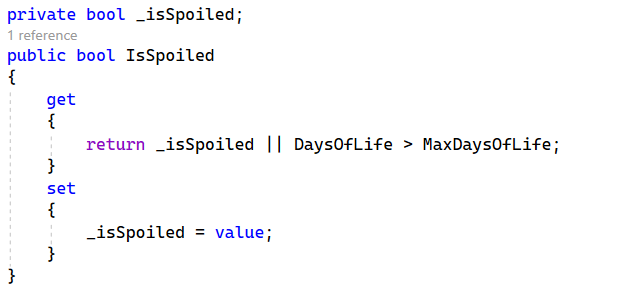
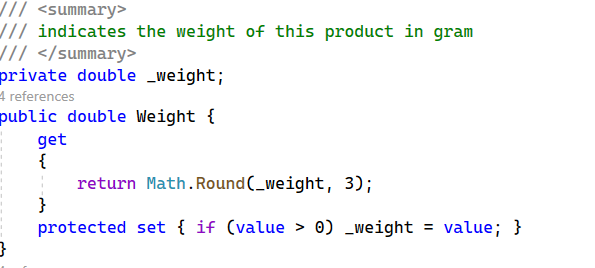
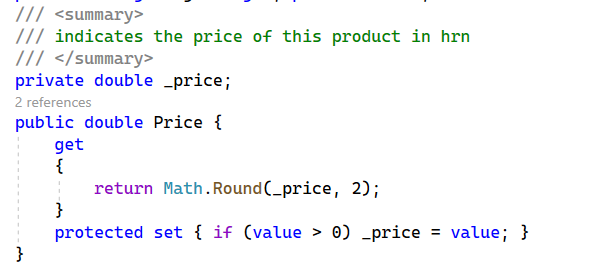
Id у класі Product є protected, аби не було доступу до нього ззовні та аби був доступ з спадкованих класів.

У клас Soup Id приватне, оскільки у даній реалізації немає спадкових класів.

Більшість полів та методів є публічними, оскільки напряму дають інформацію про об’єкт.

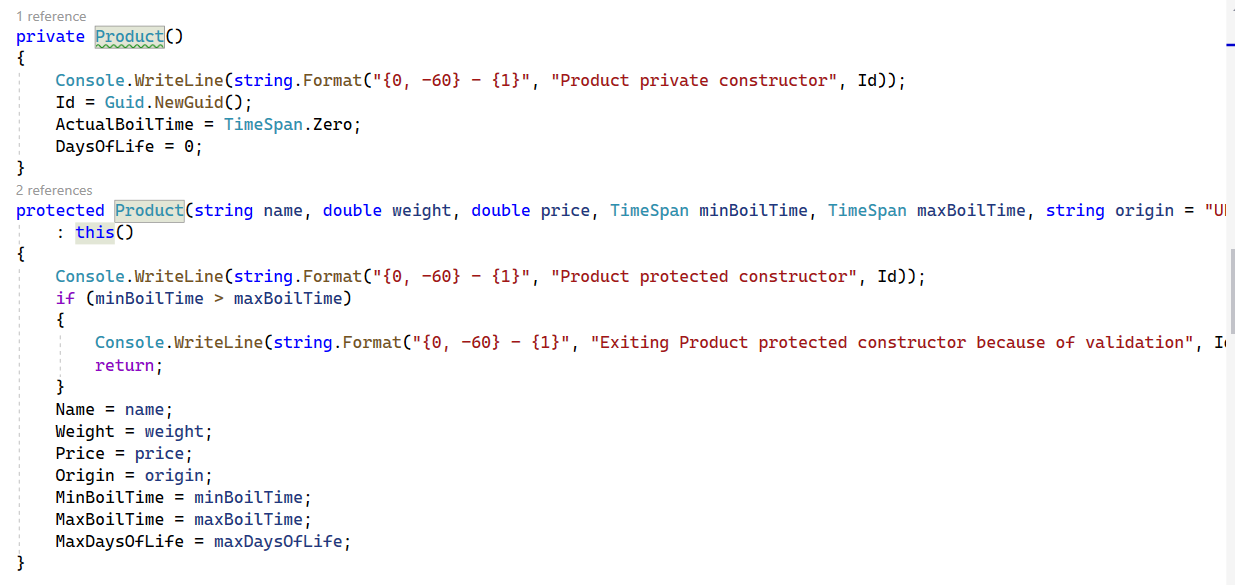
2) Приклад застосування розширених властивостей

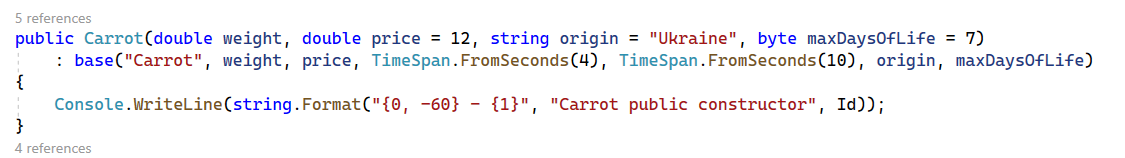


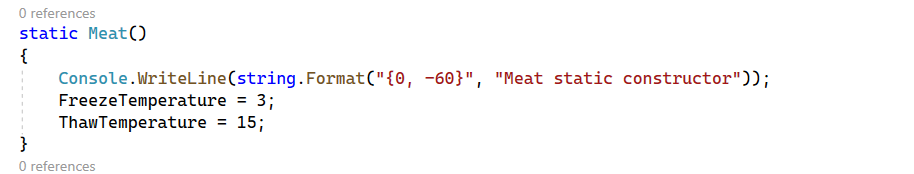
3) У наступних фотографіях з програми можна побачити конструктори:

За замовчуванням, з параметрами, приватний за замовчуванням

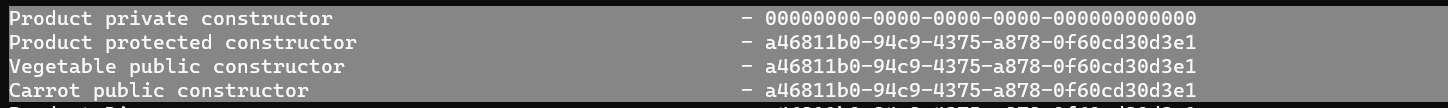




Статичний:



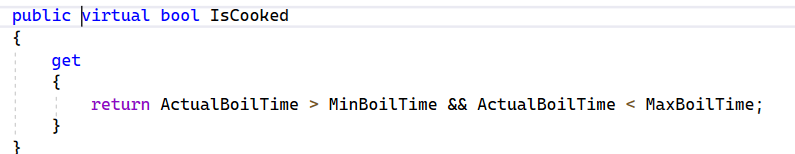
Також з наступного знімку екрана можна побачити, як викликаються конструктори у програмі:

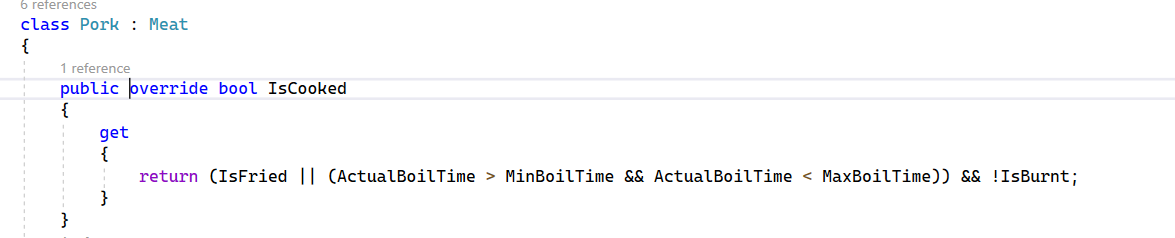


У випадку з статичним конструктором, він викликається, якщо створено об’єкт у перший раз:

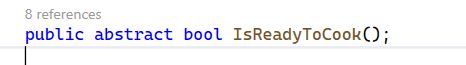


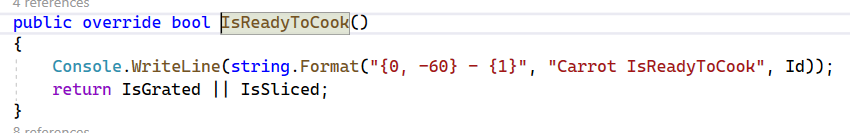
3) Для класу Product я зробив властивість virtual, аби у спадкових класах її можна було перезаписати.



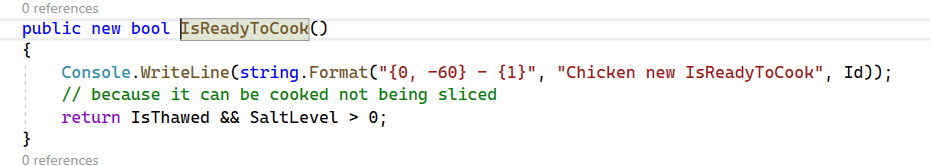


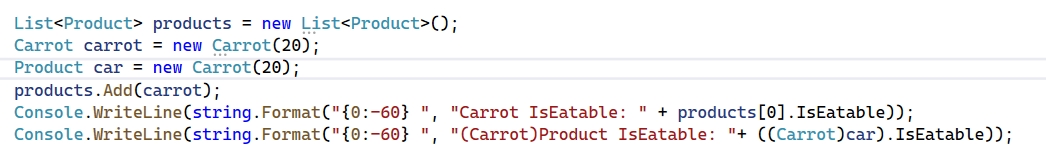
Також я додав абстрактний метод, який визначав у спадкових класах:





І також, я у класі Carrot створив новий метод, що викликався лише за умови, що змінна, що містить об’єкт класу, є Carrot, а не Vegetable або Product.

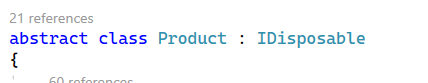
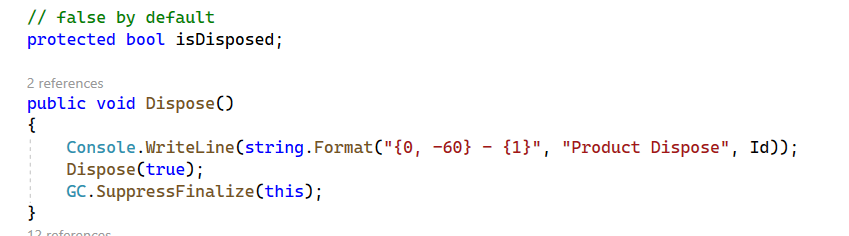


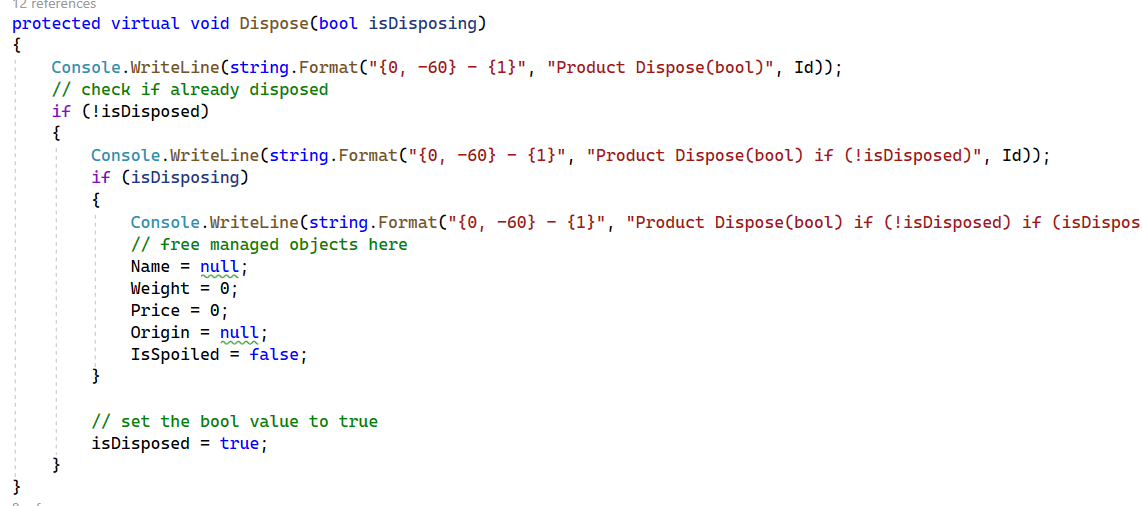
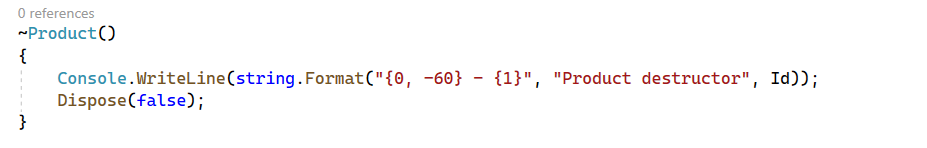




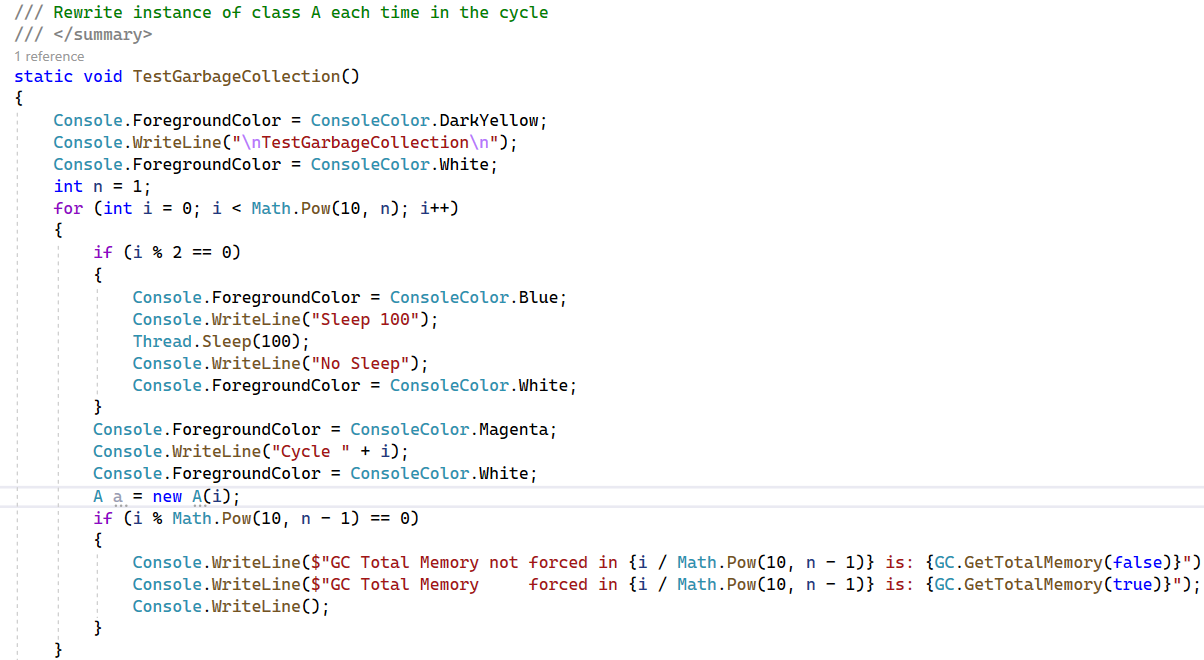
Оскільки, коли об’єкт морква є у змінній класу вищого, то викликається метод IsReadyToCook звідти.

5) За рахунок рядку GC.SuppressFinalize(this) уникається конфлікт між деструктором і IDisposable. Також, у деструкторі викликається метод Dispose(false), аби очистити об’єкт за описаним методом.

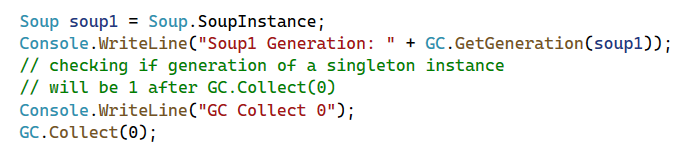
 

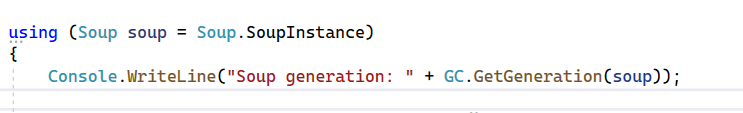
6) У даному методі можна побачити, що з другого разу починається збір GC змінної А за рахунок виклику деструктора.





Також можна помітити, як переходять об’єкти у наступну генерацію після примусового збирання GC.Collect(0);





**Висновки:**

У даній лабораторній роботі я практикувався основним принципам ООП та взаємодією з пам’яті у середовищі С#.

1. Забезпечив наявність у класах полів та методів з різними модифікаторами доступу.
2. Забезпечив наявність у класах властивостей, а також властивостей, складніших, ніж просто get; set;
3. Створив для розроблюваних класів такі конструктори:
   * Публічний конструктор за замовчанням;
   * Публічний конструктор з параметрами;
   * Захищений конструктор з параметрами;
   * Приватний конструктор;
   * Статичний конструктор.

Продемонстрував, яким чином викликаються конструктори базового та дочірнього класів.

1. Використав віртуальні та перевизначені методи, або властивості.
2. Додав до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:
   * реалізував інтерфейс IDisposable;
   * створив деструктори;
   * забезпечив уникнення конфліктів між Dispose та деструктором.
3. Забезпечив виклики методів GC таким чином, щоб можна було простежити життєвий цикл об’єктів, що обробляються (зокрема, використати методи Collect, SupressFinalize, ReRegisterForFinalize, GetTotalMemory, GetGeneration, WaitForPendingFinalizers). Створив ситуацію, яка спровокує примусове збирання сміття GC.