

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 № 1**

з дисципліни “Програмування”

тема “C# .Net. Реалізація основних принципів ООП мовою C#”

Виконав студент

II курсу групи КП-02 Щербатюк Ярослав Сергійович

Перевірив “ ” “ ” 20 р

викладач Заболотня Тетяна Миколаївна

Київ 2021

# 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 **(1 бал)**.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обґрунтувати доцільність створення властивості **(1 бал)**.
3. Створити для розроблюваних класів такі конструктори **(2 бали)**:
   * конструктор за замовчуванням;
   * конструктор з параметрами;
   * приватний конструктор;
   * статичний конструктор.

Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.

1. Використати віртуальні та перевизначені методи **(1 бал)**.
2. Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів **(2 бали)**:
3. реалізувати інтерфейс IDisposable;
4. створити деструктори;
5. забезпечити уникнення конфліктів між Dispose та деструктором.
6. 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 **(2 бали)**.

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

|  |
| --- |
|  |
| Рис. UML діаграма класів |

**Фрагменти коду**

|  |
| --- |
| Program.cs |
| using System;  namespace lab1  {      class Program      {          static void Main()          {              Pet pet = new Pet("Yaroslav Scherbatyuk", "ABCD1234");              FarmPet farmPet = new FarmPet("Yaroslav Scherbatyuk", "ABCD1234", 5);              HomePet homePet = new HomePet("Yaroslav Scherbatyuk", "ABCD1234", 300);              Dog dog = Dog.GetPet("Ivan", "OO0000OO", 450, 900);              DisposePets();              Console.WriteLine("\nMemory before creating 10\_000 obj: " + GC.GetTotalMemory(false));                for (int i = 0; i < 10\_000; i++)              {                  Pet pet1 = new Pet();                  pet1.Dispose();              }              Console.WriteLine($"Memory after creating 10\_000 obj: {GC.GetTotalMemory(false)}\n");              Console.WriteLine($"Generation of 'dog' obj before collecting: {GC.GetGeneration(dog)}\n");              GC.Collect(2);              GC.WaitForPendingFinalizers();                Console.WriteLine($"\nMemory after collecting garbage: {GC.GetTotalMemory(false)}\n");              Console.WriteLine($"Generation of 'dog' obj after collecting: {GC.GetGeneration(dog)}");              Console.WriteLine($"Garbage Collection count: {GC.CollectionCount(0)}\n");          }          private static void DisposePets()          {              Horse horse = Horse.GetPet();              Dog dog = Dog.GetPet();              horse.Dispose();              dog.Dispose();              GC.ReRegisterForFinalize(horse);              GC.ReRegisterForFinalize(dog);          }      }  } |

|  |
| --- |
| Pet.cs |
| using System;  using static System.Console;  namespace lab1  {      class Pet : IDisposable      {          protected string ownerName;          protected string licenceID;          protected static string petType;          protected bool disposed;          public Pet()          {              this.ownerName = "noOwner";              this.licenceID = "OOOO0000";          }          public Pet(string ownerName, string licenceID)          {              this.ownerName = ownerName;              this.licenceID = licenceID;          }          public virtual string GetPetDescription()          {              return $"Pet description:\n - Name of pet owner: {ownerName}\n - Licence ID: {licenceID}";          }          public virtual void Dispose()          {              CleanUp(true);              GC.SuppressFinalize(this);          }          protected void CleanUp(bool disposing)          {              if (!this.disposed)              {                  if (disposing)                  {                      // Disposing managed resources                  }                  // Disposing unmanaged resources                  disposed = true;              }          }          ~Pet()          {              CleanUp(false);              WriteLine("- Pet destructor called -");          }      }  } |

|  |
| --- |
| HomePet.cs |
| using System;  using static System.Console;  namespace lab1  {      class HomePet : Pet      {          protected double amountOfFoodToBeFull;          protected double currentAmountOfFoodNeeded;          public HomePet(string ownerName, string licenceID, double amountOfFoodToBeFull) : base(ownerName, licenceID)          {              if (amountOfFoodToBeFull > 0)              {                  this.amountOfFoodToBeFull = amountOfFoodToBeFull;              }              else              {                  this.amountOfFoodToBeFull = 0;              }          }          public HomePet() : base()          {              this.amountOfFoodToBeFull = 500;              petType = "Lives in a Home";          }          public void FeedThePet(double foodAmount)          {              if (currentAmountOfFoodNeeded > foodAmount)              {                  currentAmountOfFoodNeeded -= foodAmount;              }              else              {                  currentAmountOfFoodNeeded = 0;              }          }          public override string GetPetDescription()          {              return $"Pet description:\n - Name of pet owner: {ownerName}\n - Licence ID: {licenceID}\n - Type: {petType}\n - Amount of food to be full: {amountOfFoodToBeFull} grams";          }          ~HomePet()          {              WriteLine("- HomePet destructor called -");          }      }  } |

|  |
| --- |
| FarmPet.cs |
| using System;  using static System.Console;  namespace lab1  {      class FarmPet : Pet      {          protected double amountOfWaterToOvercomeThirst;          protected double currentAmountOfWaterNeeded;          public FarmPet(string ownerName, string licenceID, double amountOfWaterPerDay) : base(ownerName, licenceID)          {              if (amountOfWaterPerDay > 0)              {                  this.amountOfWaterToOvercomeThirst = amountOfWaterPerDay;              }              else              {                  this.amountOfWaterToOvercomeThirst = 0;              }          }          public FarmPet() : base()          {              this.amountOfWaterToOvercomeThirst = 0;              petType = "Lives in a Farm";          }          public void FeedThePet(double waterAmount)          {              if (currentAmountOfWaterNeeded > waterAmount)              {                  currentAmountOfWaterNeeded  -= waterAmount;              }              else              {                  currentAmountOfWaterNeeded  = 0;              }          }          public override string GetPetDescription()          {              return $"Pet description:\n - Name of pet owner: {ownerName}\n - Licence ID: {licenceID}\n - Type: {petType}\n - Amount of water to not be thirst: {amountOfWaterToOvercomeThirst} grams";          }          ~FarmPet()          {              WriteLine("- FarmPet destructor called -");          }      }  } |

|  |
| --- |
| Dog.cs |
| using System;  using static System.Console;  namespace lab1  {      class Dog : HomePet      {          private DateTime birthDate;          private double consumptionPerDay;          private Dog() : base()          {              this.birthDate = DateTime.Now;              this.consumptionPerDay = 0;          }          private Dog(string ownerName, string licenceID, double amountOfFoodToBeFull, double consumptionPerDay) : base(ownerName, licenceID, amountOfFoodToBeFull)          {              this.birthDate = DateTime.Now;              this.consumptionPerDay = consumptionPerDay;          }          public static Dog GetPet(string name, string licenceID, double amountOfFoodToBeFull, double consumptionPerDay)          {              return new Dog(name, licenceID, amountOfFoodToBeFull, consumptionPerDay);          }          public static Dog GetPet()          {              return new Dog();          }          public double ConsumptionPerDay          {              get              {                  return consumptionPerDay;              }              set              {                  if (value > 0)                  {                      this.consumptionPerDay = value;                  }                  else                  {                      this.consumptionPerDay = 0;                  }              }          }          public double Meals          {              get              {                  if (ConsumptionPerDay != 0)                  {                      return ConsumptionPerDay / amountOfFoodToBeFull;                  }                  else                  {                      return -1;                  }              }          }          public void GiveTheCommandToGiveAPaw (int pawNumber)          {              if(pawNumber == 1)              {                  WriteLine("Dog gives you left paw!");              }              else if(pawNumber == 2)              {                  WriteLine("Dog gives you right paw!");              }              else              {                  WriteLine("Dog doesn't have paws with number like that!");              }          }          public override string GetPetDescription()          {              return $"Pet description:\n - Owner: {ownerName}\n - Licence ID: {licenceID}" +              $"\n - Type: {petType}\n - Amount of food to be full: {amountOfFoodToBeFull}" +              $"\n - Birth Date: {birthDate.ToString("d")}";          }          ~Dog()          {              WriteLine("- Dog destructor called -");          }      }  } |

|  |
| --- |
| Horse.cs |
| using System;  using static System.Console;  namespace lab1  {      class Horse : FarmPet      {          private DateTime birthDate;          private double consumptionPerDay;          private Horse() : base()          {              this.birthDate = DateTime.Now;              this.consumptionPerDay = 0;          }          private Horse(string ownerName, string licenceID, double amountOfWaterToOvercomeThirst, double consumptionPerDay) : base(ownerName, licenceID, amountOfWaterToOvercomeThirst)          {              this.birthDate = DateTime.Now;              this.consumptionPerDay = consumptionPerDay;          }          public static Horse GetPet(string name, string licenceID, double amountOfWaterToOvercomeThirst, double consumptionPerDay)          {              return new Horse(name, licenceID, amountOfWaterToOvercomeThirst, consumptionPerDay);          }          public static Horse GetPet()          {              return new Horse();          }          public double ConsumptionPerDay          {              get              {                  return consumptionPerDay;              }              set              {                  if (value > 0)                  {                      this.consumptionPerDay = value;                  }                  else                  {                      this.consumptionPerDay = 0;                  }              }          }          public double Drinks          {              get              {                  if (ConsumptionPerDay != 0)                  {                      return ConsumptionPerDay / amountOfWaterToOvercomeThirst;                  }                  else                  {                      return -1;                  }              }          }          public void StrokeTheHorse()          {              WriteLine("Horse is happy that you stroked her! Horse love u <3");          }          public override string GetPetDescription()          {              return $"Horse description:\n - Owner: {ownerName}\n - Licence ID: {licenceID}" +              $"\n - Type: {petType}\n - Amount of water to not be thirst: {amountOfWaterToOvercomeThirst}" +              $"\n - Birth Date: {birthDate.ToString("d")}";          }          ~Horse()          {              WriteLine("- Dog destructor called -");          }      }  } |

|  |
| --- |
| Memory before creating 10\_000 obj: 77480  Memory after creating 10\_000 obj: 490552  Generation of 'dog' obj before collecting: 0  - Dog destructor called -  - FarmPet destructor called -  - Pet destructor called -  - Dog destructor called -  - HomePet destructor called -  - Pet destructor called -  Memory after collecting garbage: 87384  Generation of 'dog' obj after collecting: 1  Garbage Collection count: 1 |

**Висновки**

В результаті виконання лабораторної роботи відбулось ознайомлення з основами об’єктно-орієнтованого підходу програмування при створенні різного ПЗ на мові С#, створенням клаcів, об’єктів класів, механізмами інкапсуляції, наслідування та поліморфізму. Були випробувані механізми управління ресурсами, реалізованих у платформі .Net.