

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 Товстига Максим Анатолійович

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:

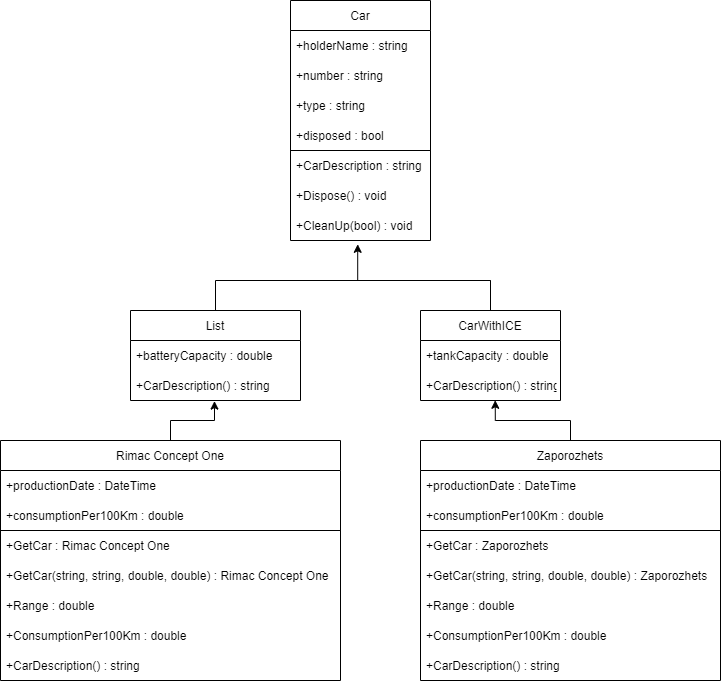
1. Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір **(1 бал)**.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обґрунтувати доцільність створення властивості **(1 бал)**.
3. Створити для розроблюваних класів такі конструктори **(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.

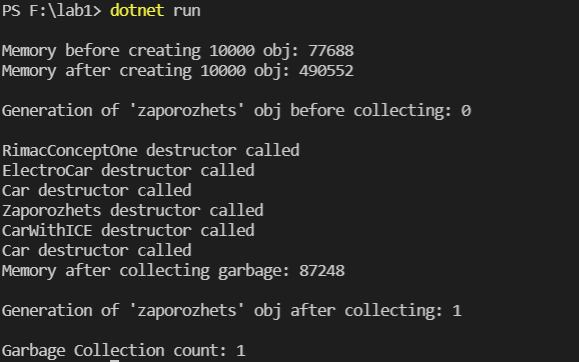
# UML діаграма класів

****

**Текст коду програми**

|  |
| --- |
| **Program.cs** |
| using System;  using static System.Console;      class Program      {          static void Main()          {              Car car = new Car("Arthurian", "CA1337CA");              ElectroCar eCar = new ElectroCar("Yarichelo", "CA2288FF", 10000);              CarWithICE iceCar = new CarWithICE("Maximus", "BH0000AK", 60);              Zaporozhets zaporozhets = Zaporozhets.GetCar("Andrew", "AA1111EE", 70, 4);              DisposeCars();              WriteLine("\nMemory before creating 10000 obj: " + GC.GetTotalMemory(false));                for (int i = 0; i < 10000; i++)              {                  Car car1 = new Car();                  car1.Dispose();              }              WriteLine($"Memory after creating 10000 obj: {GC.GetTotalMemory(false)}\n");              WriteLine($"Generation of 'zaporozhets' obj before collecting: {GC.GetGeneration(zaporozhets)}\n");              GC.Collect(2);              GC.WaitForPendingFinalizers();                WriteLine($"Memory after collecting garbage: {GC.GetTotalMemory(false)}\n");              WriteLine($"Generation of 'zaporozhets' obj after collecting: {GC.GetGeneration(zaporozhets)}\n");              WriteLine($"Garbage Collection count: {GC.CollectionCount(0)}");          }          private static void DisposeCars()          {              RimacConceptOne rimac = RimacConceptOne.GetCar();              rimac.Dispose();              GC.ReRegisterForFinalize(rimac);              Zaporozhets zaporozhets = Zaporozhets.GetCar();              zaporozhets.Dispose();              GC.ReRegisterForFinalize(zaporozhets);          }      } |
| **Car.cs** |
| using System;  using static System.Console;      class Car : IDisposable      {          protected string number;          protected string holderName;          protected static string type;          protected bool disposed;          public Car()          {              this.number = "unknown";              this.holderName = "unknown";          }          public Car(string name, string number)          {              this.number = number;              this.holderName = name;          }          public virtual string GetCarDescription()          {              return $"Car description: Holder - {holderName},\n Number - {number}";          }          public virtual void Dispose()          {              CleanUp(true);              GC.SuppressFinalize(this);          }          protected void CleanUp(bool disposing)          {              if (!this.disposed)              {                  if (disposing)                  {}                  disposed = true;              }          }          ~Car()          {              CleanUp(false);              WriteLine("Car destructor called");          }      } |
| **CarWithICE.cs** |
| using System;  using static System.Console;      class CarWithICE : Car      {          protected double maxTankCapacity;          protected double currentFuelLevel = 0;          static CarWithICE()          {              type = "ICE Car";          }          public CarWithICE(string name, string number, double maxTankCapacity) : base(name, number)          {              if (maxTankCapacity > 0)              {                  this.maxTankCapacity = maxTankCapacity;              }              else              {                  this.maxTankCapacity = 0;              }          }          public CarWithICE() : base()          {              this.maxTankCapacity = 0;          }          public void FillCar(double fuelAmount)          {              if ((maxTankCapacity - currentFuelLevel) > fuelAmount)              {                  currentFuelLevel += fuelAmount;              }              else              {                  currentFuelLevel = maxTankCapacity;              }          }          public override string GetCarDescription()          {              return $"Car description: Holder: {holderName},\n Number: {number}],\n Type: {type},\n Tank capacity: {maxTankCapacity} Litres";          }            ~CarWithICE()          {              WriteLine("CarWithICE destructor called");          }      } |
| **ElectroCar.cs** |
| using System;  using static System.Console;      class ElectroCar : Car      {          protected double maxBatteryCapacity;          protected double currentChargeLevel = 0;          static ElectroCar()          {              type = "Electro Car";          }          public ElectroCar(string name, string number, double maxBatteryCapacity) : base(name, number)          {              if (maxBatteryCapacity > 0)              {                  this.maxBatteryCapacity = maxBatteryCapacity;              }              else              {                  this.maxBatteryCapacity = 0;              }          }          public ElectroCar() : base()          {              this.maxBatteryCapacity = 0;          }          public void ChargeCar(double chargeAmount)          {              if ((maxBatteryCapacity - currentChargeLevel) > chargeAmount)              {                  currentChargeLevel += chargeAmount;              }              else              {                  currentChargeLevel = maxBatteryCapacity;              }          }          public override string GetCarDescription()          {              return $"Car description: Holder: {holderName},\n Number: {number},\n Type: {type},\n Battery capacity: {maxBatteryCapacity} Watt\*Hour";          }            ~ElectroCar()          {              WriteLine("ElectroCar destructor called");          }      } |
| **RimacConceptOne.cs** |
| using System;  using static System.Console;      class RimacConceptOne : ElectroCar      {          private DateTime productionDate;          private double expensePer100Km;          private RimacConceptOne() : base()          {              this.productionDate = DateTime.Now;              this.expensePer100Km = 0;          }          private RimacConceptOne(string name, string number, double maxBatteryCapacity, double expense) : base(name, number, maxBatteryCapacity)          {              this.productionDate = DateTime.Now;                this.expensePer100Km = expense;          }          public static RimacConceptOne GetCar(string name, string number, double maxBatteryCapacity, double expense)          {              return new RimacConceptOne(name, number, maxBatteryCapacity, expense);          }          public static RimacConceptOne GetCar()          {              return new RimacConceptOne();          }          public double Range          {              get              {                  if (expensePer100Km != 0)                  {                      return maxBatteryCapacity / expensePer100Km;                  }                  else                  {                      return -1;                  }              }          }          public double ExpensePer100Km          {              get              {                  return expensePer100Km;              }              set              {                  if (value > 0)                  {                      this.expensePer100Km = value;                  }                  else                  {                      this.expensePer100Km = 0;                  }              }          }          public void RemoteOpenDoor()          {              WriteLine($"Door opened");          }          public override string GetCarDescription()          {              return $"Car description: - Holder: {holderName} - Number: {number} -\n" +              $"- Type: {type} - Battery capacity: {maxBatteryCapacity} Watt \* Hour -\n" +              $"- Production Date: {productionDate.ToString("d")} -";          }          ~RimacConceptOne()          {              WriteLine("RimacConceptOne destructor called");          }      } |
| **Zaporozhets.cs** |
| using System;  using static System.Console;      class Zaporozhets : CarWithICE      {          private DateTime productionDate;          private double expensePer100Km;          private Zaporozhets() : base()          {              this.productionDate = DateTime.Now;              this.expensePer100Km = 0;          }          private Zaporozhets(string name, string number, double tankCapacity, double expense) : base(name, number, tankCapacity)          {              this.productionDate = DateTime.Now;              this.expensePer100Km = expense;          }          public static Zaporozhets GetCar(string name, string number, double tankCapacity, double expense)          {              return new Zaporozhets(name, number, tankCapacity, expense);          }          public static Zaporozhets GetCar()          {              return new Zaporozhets();          }          public double Range          {              get              {                  if (expensePer100Km != 0)                  {                      return maxTankCapacity / expensePer100Km;                  }                  else                  {                      return -1;                  }              }          }          public double ExpensePer100Km          {              get              {                  return expensePer100Km;              }              set              {                  if (value > 0)                  {                      this.expensePer100Km = value;                  }                  else                  {                      this.expensePer100Km = 0;                  }              }          }          public void OpenTrunk()          {              WriteLine("Trunk opened");          }          public override string GetCarDescription()          {              return $"Car description: - Holder: {holderName},\n Number: {number} \n" +              $"Type: {type}\n Tank Capacity: {maxTankCapacity} Litres\n" +              $"Production Date: {productionDate.ToString("d")}";          }            ~Zaporozhets()          {              WriteLine("Zaporozhets destructor called");          }      } |

**Результати**



**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.