

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 " Реалізація основних принципів ООП "

|  |  |  |
| --- | --- | --- |
| Виконав  студент 2 курсу  групи КП-01  Беліцький Олександр Сергійович  (*прізвище, ім’я, по батькові*) |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

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 завдання**

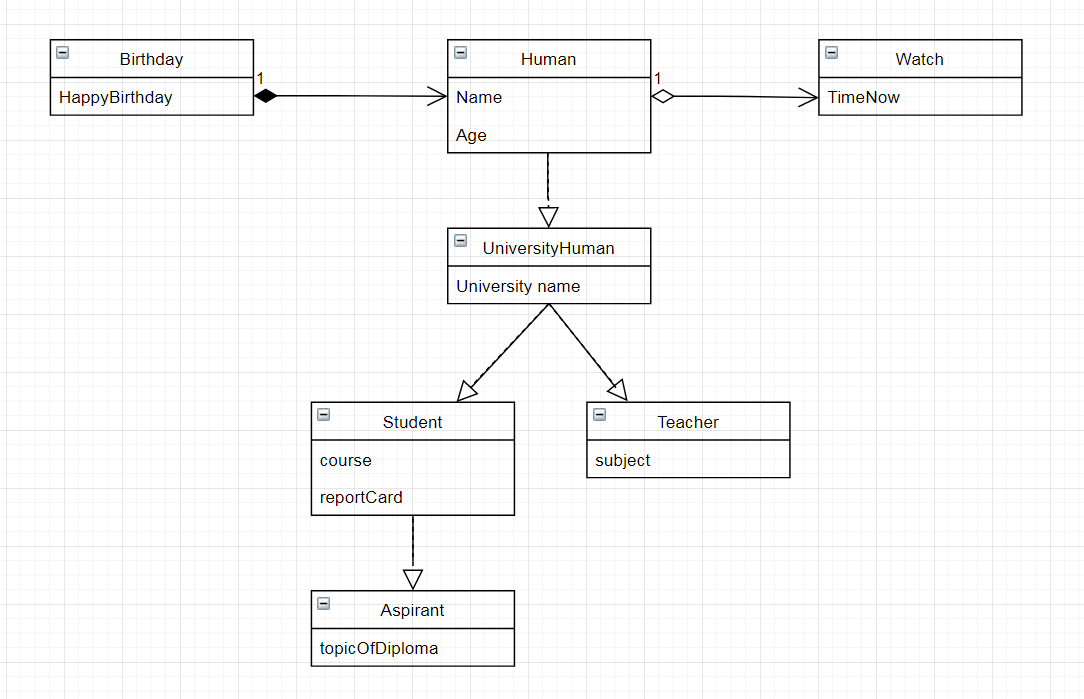
Побудувати ієрархію класів, що відтворюватимуть відношення наслідування між об’єктами реального світу (кількість класів >= 5). При цьому:

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.

**Аналіз вимог і проектування**

*****рис.1 UML-діаграма класів*

Зв'язки:

День народження - Людина - композиція, бо День народження без людини існувати не може.

День народження - "Годинник" - агрегація, бо час існує окремо від людини і одночасно з нею.

Людина - Людина з університету - наслідування, бо людей багато і піпгрупа людей, що причетні до університету.

Людина з університету - Студент - Аспірант - наслідування(по аналогії з попереднім).

Людина з університету - Вчитель - наслідування(по аналогії з попереднім).

**Тексти коду програм**

|  |
| --- |
| **Program.cs** |
| using System;  using static System.Console;  class Program  {  static void Main(string[] args)  {  Human h = new Human("Gerd", 44);  h.Print();  Birthday.HappyBirthday(h);  WriteLine("-----\n");  UniversityPerson universityPerson= new UniversityPerson("Nun", 33, "ZhTI");  universityPerson.GoToClass();  WriteLine("-----\n");  Teacher teacher = new Teacher("Harley", 43, "KPI", "PE");  teacher.GoToClass();  WriteLine("-----\n");      Student s = new Student("Bob", 22, "KPI", 4, "ssd23w");  s.Print();  WriteLine();    s.GoToClass();  WriteLine();  Birthday.HappyBirthday(s);  WriteLine("-----\n");  Aspirant a = new Aspirant("Huan", 62, "VNTU", 4, "323sd2", "Football");  a.Print();  WriteLine("-----\n");      Watch watch = new Watch();  WriteLine("-----\n");  GCProcess();  }  static void GCProcess()  {  Human[] group = new Human[1500];  for(int i = 0; i < 1350; i++)  {  group[i] = new Human();  }  WriteLine("Total memory before: {0}", GC.GetTotalMemory(false));    GC.Collect();  GC.WaitForPendingFinalizers();  WriteLine("Total memory after: {0}", GC.GetTotalMemory(false));  }  }  public class Human : IDisposable  {  private string name;  private int age;  public string Name  {  set  {  if (!string.IsNullOrEmpty(value) && !string.IsNullOrWhiteSpace(value))  name = value;  }  get { return name; }  }  public int Age  {  set  {  if (value > 0 && value < 150)  age = value;  }  get { return age; }  }  public Human() { }  public Human(string name, int age)  {  this.name = name;  this.age = age;  WriteLine("I'm a human");  }  public virtual void Print()  {  WriteLine($"Name = {this.Name}");  WriteLine($"Age = {this.age}");  }  private bool disposed = false;  ~Human()  {  Dispose(false);  Beep();  WriteLine("Destructor has worked");  }  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  Beep();  WriteLine("IDisposable method has worked");  }  protected virtual void Dispose(bool disposing)  {  if(!disposed)  {  if(disposing)  {  WriteLine("delete managed resourses");  }  disposed = true;  WriteLine("delete unmanaged resourses");  }  }  }  public class UniversityPerson : Human  {  protected string UniversityName;  public UniversityPerson(string name, int age, string UniversityName) : base(name, age)  {  this.UniversityName = UniversityName;  WriteLine("I'm university person");  }  //TODO: GoToClass  public virtual void GoToClass()  {  WriteLine("I'm going to classroom");  }  }  public sealed class Teacher : UniversityPerson  {  private string subject;  public Teacher(string name, int age, string UniversityName, string subject) : base(name, age, UniversityName)  {  this.subject = subject;  WriteLine("I'm a teacher");  }  public override void GoToClass()  {  base.GoToClass();  WriteLine("I'm teaching here");  }  }  public class Student : UniversityPerson  {  protected int course;  protected string reportCard;  public Student(string name, int age, string UniversityName, int course, string reportCard) : base(name, age, UniversityName)  {  this.course = course;  this.reportCard = reportCard;  WriteLine("I'm a student");  }  public int Course  {  get { return course; }  set { course = value; }  }  public string GradeBook  {  get { return reportCard; }  set { reportCard = value; }  }  public override void Print()  {  WriteLine("The values of fields are:");  WriteLine($"Name = {this.Name}");  WriteLine($"Age = {this.Age}");  WriteLine($"Course = {course}");  WriteLine($"GradeBook = {reportCard}");  }  public override void GoToClass()  {  base.GoToClass();  WriteLine("I'm studying here");  }  }  public class Aspirant : Student  {  protected string topic;  public Aspirant(string name, int age, string UniversityName, int course, string reportCard, string topic) : base(name, age, UniversityName, course, reportCard)  {  base.Name = name;  this.course = course;  this.reportCard = reportCard;  this.topic = topic;  WriteLine("I'm aspirant");  }  public new void Print()  {  base.Print();  WriteLine($"Topic = {topic}");  }  }  public class Birthday  {  private Birthday() { }  public static void HappyBirthday(Human human)  {  human.Age = human.Age + 1;  WriteLine("{0} is {1} years old now", human.Name, human.Age);  }  }  public class Watch  {  static DateTime time;  static Watch()  {  time = DateTime.Now;    WriteLine("The evolution of man is now at : {0}", time);  }  } |

**Приклади результатів**

1. *Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір.*

У даній роботі були використані 3 типи модификаторів доступу: public, private, protected.

Public методи та поля були створені для зовнішньої взаємодії.

Private поля були створені для забезпечення вищого рівня безпеки та відгородити користувача від прямого впливу на ці об'єкти.

Protected було встановлено на поля, що мали бути використані у класах наслідниках.

1. *Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості.*

Властивості були використані для зменшення кількості коду і кращої читабельності (замість двох методів на отримання і встановлення значень з перевірками використана одна властивість).

Приклад властивостей із класу *Human.*

|  |
| --- |
| public string Name  {  set  {  if (!string.IsNullOrEmpty(value) && !string.IsNullOrWhiteSpace(value))  name = value;  }  get { return name; }  }  public int Age  {  set  {  if (value > 0 && value < 150)  age = value;  }  get { return age; }  } |

1. *Створити для розроблюваних класів такі конструктори:*

• *конструктор за замовчанням*;(приклад з класу *Human*)

|  |
| --- |
| public Human() { } |

• *конструктор з параметрами*; (приклад з класу *Human*)

|  |
| --- |
| public Human(string name, int age)  {  this.name = name;  this.age = age;  WriteLine("I'm a human");  } |

• *приватний конструктор;* (приклад з класу *Birthday*)

|  |
| --- |
| private Birthday() { } |

• *статичний конструктор.* (приклад з класу *Watch*)

|  |
| --- |
| static Watch()  {  time = DateTime.Now;    WriteLine("The evolution of man is now at : {0}", time);  } |

*Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.*

Приклад створення сутності Аспіранта:

|  |
| --- |
|  |

1. *Використати віртуальні та перевизначені методи.*

Було створено віртуальний метод *Print,* який був перевизначений за допомогою ключових слів *override* та *new*.

|  |
| --- |
| public virtual void Print()  {  WriteLine($"Name = {this.Name}");  WriteLine($"Age = {this.age}");  }  //Клас Human |
| public override void Print()  {  WriteLine("The values of fields are:");  WriteLine($"Name = {this.Name}");  WriteLine($"Age = {this.Age}");  WriteLine($"Course = {course}");  WriteLine($"GradeBook = {reportCard}");  }  // Клас Student |
| public new void Print()  {  base.Print();  WriteLine($"Topic = {topic}");  }  // Клас Aspirant |

1. *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.*

Все зазначене забезпечено у наступному фрагменті коду:

|  |
| --- |
| private bool disposed = false;  ~Human()  {  Dispose(false);  Beep();  WriteLine("Destructor has worked");  }  public void Dispose()  {  Dispose(true);  GC.SuppressFinalize(this);  Beep();  WriteLine("IDisposable method has worked");  }  protected virtual void Dispose(bool disposing)  {  if(!disposed)  {  if(disposing)  {  WriteLine("delete managed resourses");  }  disposed = true;  WriteLine("delete unmanaged resourses");  }  } |

1. *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.*

Цей проєкт не є дуже затратним у пам'яті, тому тут немає необхідності збору сміття та очищення un-/managed ресурсів. Тому, ми зімітуємо таку ситуацію:

|  |
| --- |
| static void GCProcess()  {  Human[] group = new Human[1500];  for(int i = 0; i < 1350; i++)  {  group[i] = new Human();  }  WriteLine("Total memory before: {0}", GC.GetTotalMemory(false));    GC.Collect();  GC.WaitForPendingFinalizers();  WriteLine("Total memory after: {0}", GC.GetTotalMemory(false));  } |

**Висновки**

Виконавши дану лабораторну роботу було підкоректовано знання з основ об'єктно-орієнтовного програмування. Були набути навички з створення деструкторів, імплементації інтерфейсу IDisposiable та усунення конфлікту між цими структурами.

Також ознайомився з класом System.GC. Найбільш поширеним випадком його використання є збір сміття при роботі з некерованими ресурсами, при інтенсивному виділенні великих обсягів пам'яті, при яких необхідно таке ж швидке їх звільнення.

Відкрив для себе приватний конструктор. Зазвичай він використовується в класах, що містять тільки статичні елементи. Якщо в класі один або кілька закритих конструкторів і жодного відкритого конструктора, то інші класи не зможуть створювати екземпляри цього класу.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.