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 1. Об'єктно-орієнтоване програмування та шаблони проєктування”

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

Виконала
студентка ІІ курсу
групи КП-03
Сіренко Вікторія Юріївна
(*прізвище, ім’я, по батькові*)

Перевірила
“____” “_____” 20___ р.
викладачка
Заболотня Тетяна Миколаївна
(*прізвище, ім’я, по батькові*)

Київ 2021

Мета роботи

Ознайомитися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові C#, 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:

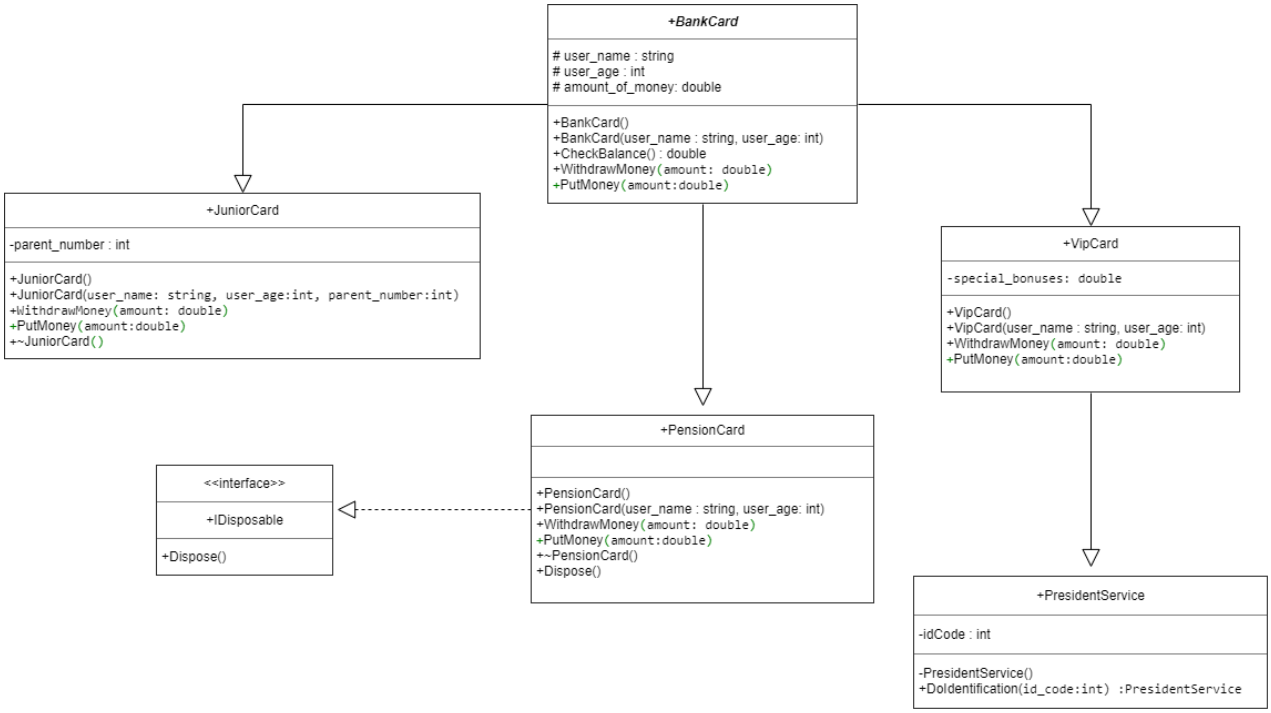
1. Забезпечити наявність у класах полів та методів з різними модифікаторами доступу, пояснити свій вибір.
2. Забезпечити наявність у класах властивостей: складніше, ніж просто get;set;, обгрунтувати доцільність створення властивості.
3. Створити для розроблюваних класів такі конструктори :
    - конструктор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

    Продемонструвати, яким чином викликаються конструктори базового та дочірнього класів.
4. Використати віртуальні та перевизначені методи.
5. Додати до класів методи, наявність яких дозволятиме управляти знищенням екземплярів цих класів:
    a. реалізувати інтерфейс IDisposable;
    b. створити деструктори;
    c. 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.

Аналіз вимог і проектування

UML діаграма класів:



Тексти коду програм

Program.cs

```
1.  using System;
2.
3.  namespace lab1
4.  {
5.      class Program
6.      {
7.          static void Main(string[] args)
8.          {
9.
10.             PresidentService presidentCard = PresidentService.Identification(1567832);
11.             Console.ReadKey();
12.             Console.Clear();
13.
14.
15.             for (int i = 0; i <= 10; i++)
16.             {
17.                 JuniorCard juniorCard = new JuniorCard();
18.                 Console.WriteLine();
19.             }
20.             GC.Collect(0, GCCollectionMode.Forced);
21.             GC.WaitForPendingFinalizers();
22.             Console.ReadKey();
23.             Console.Clear();
24.
25.
26.
27.             using (PensionCard pensionCard = new PensionCard("Sara", 65))
28.             {
29.                 GC.SuppressFinalize(pensionCard);
30.                 pensionCard.PutMoney(50);
31.                 Console.WriteLine($"Current amount of money
    {pensionCard.CheckBalance()}");
32.             }
33.             Console.ReadKey();
34.             Console.Clear();
35.
36.
37.
38.             VipCard vipCard= new VipCard("Petro", 14);
39.             Console.ReadKey();
40.             Console.Clear();
41.         }
42.     }
43.
44.
45.     public abstract class BankCard
46.     {
47.         protected string user_name;
48.         protected int user_age;
49.         protected double amount_of_money;
50.
51.
52.         public BankCard(string user_name, int user_age)
53.         {
54.             this.Name = user_name;
55.             this.Age = user_age;
56.             this.amount_of_money = default;
```

```
57.             Console.WriteLine("BankCard constructor with param");
58.         }
59.
60.
61.         public BankCard()
62.         {
63.             this.Name = "Client";
64.             this.Age = 18;
65.             this.amount_of_money = default;
66.             Console.WriteLine("BankCard constructor without param");
67.
68.         }
69.
70.         public string Name
71.         {
72.             get
73.             {
74.                 return user_name;
75.             }
76.             set
77.             {
78.                 if (!string.IsNullOrEmpty(value))
79.                 {
80.                     user_name = value;
81.                 }
82.                 else
83.                 {
84.                     Console.WriteLine("You must enter your username");
85.                 }
86.             }
87.         }
88.
89.         public abstract int Age { get; set; }
90.
91.         public double CheckBalance()
92.         {
93.             return amount_of_money;
94.         }
95.
96.         public abstract void WithdrawMoney(double amount);
97.         public abstract void PutMoney(double amount);
98.     }
99.
100.
101.
102.     public class JuniorCard : BankCard
103.     {
104.         private string parent_number;
105.
106.         public JuniorCard()
107.         {
108.             this.Name = "Junior client";
109.             this.Age = 16;
110.             this.amount_of_money = default;
111.             this.parent_number = default;
112.             Console.WriteLine("JuniorCard constructor without param");
113.
114.         }
115.
116.         public JuniorCard(string user_name, int user_age, string parent_number) :
    base(user_name, user_age)
117.         {
118.             this.parent_number = parent_number;
```

```
119.             Console.WriteLine("JuniorCard constructor with param");
120.         }
121.
122.         public string ParentNumber
123.         {
124.             get
125.             {
126.                 return parent_number;
127.             }
128.         }
129.
130.
131.
132.         public override int Age
133.         {
134.             get
135.             {
136.                 return user_age;
137.             }
138.             set
139.             {
140.                 if (value >= 16 && value <= 18)
141.                 {
142.                     user_age = value;
143.                 }
144.                 else
145.                 {
146.                     Console.WriteLine("You do not fit the age of the Junior Card,
    choose another card");
147.                 }
148.             }
149.         }
150.
151.
152.
153.         public override void WithdrawMoney(double amount)
154.         {
155.             if (amount < 0 || amount > amount_of_money)
156.             {
157.                 Console.WriteLine("It is impossible to withdraw money");
158.             }
159.             else if (amount >= 3000)
160.             {
161.                 Console.WriteLine("You can't withdraw such a large amount");
162.             }
163.             else
164.             {
165.                 this.amount_of_money = amount_of_money - amount;
166.
167.             }
168.         }
169.
170.         public override void PutMoney(double amount)
171.         {
172.             if (amount < 0)
173.             {
174.                 Console.WriteLine("Set value is negative");
175.             }
176.             else if (amount_of_money + amount > 20000)
177.             {
178.                 Console.WriteLine("It is not possible to put money on the card,
    because the card has a limit of 20,000");
179.             }
```

```
            else
            {
                this.amount_of_money = amount_of_money + amount;
            }
        }

        ~JuniorCard()
        {
            Console.WriteLine("Junior card destructor done!");
        }
    }


    public class PensionCard : BankCard, IDisposable
    {

        public PensionCard()
        {
            this.Name = "Pension client";
            this.Age = 60;
            this.amount_of_money = default;
            Console.WriteLine("PensionCard constructor without param");
        }

        public PensionCard(string user_name, int user_age) : base(user_name, user_age)
        {
            Console.WriteLine("PensionCard constructor with param");
        }

        public override int Age
        {
            get
            {
                return user_age;
            }
            set
            {
                if (value >= 60 && value <= 99)
                {
                    user_age = value;
                }
                else
                {
                    Console.WriteLine("You do not fit the age of the Pension Card, choose another card");
                }
            }
        }

        public override void PutMoney(double amount)
        {
            if (amount < 0)
            {
                Console.WriteLine("Set value is negative");
            }
            else if (amount_of_money + amount > 100000)
            {
                Console.WriteLine("It is not possible to put money on the card, because the card has a limit of 100,000");
            }
            else
            {
                this.amount_of_money = amount_of_money + amount;
```

```
241.             }
242.         }
243.
244.
245.         public override void WithdrawMoney(double amount)
246.         {
247.             if (amount < 0 || amount > amount_of_money)
248.             {
249.                 Console.WriteLine("It is impossible to withdraw money");
250.             }
251.             else
252.             {
253.                 this.amount_of_money = amount_of_money - amount;
254.             }
255.         }
256.
257.         public void Dispose()
258.         {
259.             GC.SuppressFinalize(this);
260.             Console.WriteLine("Pension card dispose done!");
261.         }
262.
263.         ~PensionCard()
264.         {
265.             Console.WriteLine("Pension card destructor done!");
266.         }
267.
268.     }
269.
270.     public class VipCard : BankCard
271.     {
272.         private double special_bonuses;
273.         static VipCard()
274.         {
275.             Console.WriteLine("Congratulations, you are the first VIP user.(static
    constructor)");
276.         }
277.
278.
279.         public VipCard()
280.         {
281.             this.Name = "Vip client";
282.             this.Age = 18;
283.             this.amount_of_money = default;
284.             this.special_bonuses = default;
285.             Console.WriteLine("VipCard constructor without param");
286.         }
287.
288.
289.         public VipCard(string user_name, int user_age) : base(user_name, user_age)
290.         {
291.             this.special_bonuses = default;
292.             Console.WriteLine("VipCard constructor with param");
293.         }
294.
295.
296.         public double Bonuses
297.         {
298.             get
299.             {
300.                 return special_bonuses;
301.             }
302.         }
```

```
303.
304.
305.          public override int Age
306.          {
307.              get
308.              {
309.                  return user_age;
310.              }
311.              set
312.              {
313.                  if (value >= 18 && value <= 99)
314.                  {
315.                      user_age = value;
316.                  }
317.                  else
318.                  {
319.                      Console.WriteLine("You do not fit the age of the Vip Card, choose
    another card");
320.                  }
321.              }
322.          }
323.
324.
325.          public override void PutMoney(double amount)
326.          {
327.              if (amount < 0)
328.              {
329.                  Console.WriteLine("Set value is negative");
330.              }
331.              else
332.              {
333.                  this.amount_of_money = amount_of_money + amount;
334.                  if (amount >= 50000)
335.                  {
336.                      special_bonuses += 5;
337.                  }
338.              }
339.          }
340.
341.
342.          public override void WithdrawMoney(double amount)
343.          {
344.              if (amount < 0 || amount > amount_of_money)
345.              {
346.                  Console.WriteLine("It is impossible to withdraw money");
347.              }
348.              else if (amount > 100000)
349.              {
350.                  Console.WriteLine("We need to make sure that you are withdrawing
    money, so you will now be called to confirm your identity");
351.              }
352.              else
353.              {
354.                  this.amount_of_money = amount_of_money - amount;
355.              }
356.          }
357.      }
358.
359.
360.      public class PresidentService : VipCard
361.      {
362.          private static int idCode = 1567832;
363.          private PresidentService()
```

```
364.         {
365.             Console.WriteLine("President service. This is private constructor");
366.         }
367.
368.         public static PresidentService Identification(int id_code)
369.         {
370.             if (id_code == idCode)
371.             {
372.                 PresidentService presidentCard = new PresidentService();
373.                 return presidentCard;
374.             }
375.             Console.WriteLine("You entered an incorrect id");
376.             return null;
377.         }
378.
379.
380.     }
381.
382. }
```

Висновки

Під час виконання лабораторної роботи №1, ми ознайомилися з основами об’єктно-орієнтованого підходу до створення ПЗ у мові C#, створенням класів, об’єктів, механізмів інкапсуляції, наслідування та поліморфізму.

Інкапсуляція - це механізм програмування, який об'єднує разом код і дані, якими він маніпулює, виключаючи як втручання ззовні, так і неправильне використання даних.
Тобто інкапсуляція надає можливість приховати зайві деталі реалізації від користувача об'єкта.

Наслідування – це один з принципів об’єктно-орієнтованого програмування, який дає можливість класу використовувати програмний код іншого (базового) класу, доповнюючи його своїми власними деталями реалізації.

Поліморфізм у C# - це здатність об’єктів різних типів надавати унікальний інтерфейс для різних реалізацій методів.
У C# поліморфізм реалізується шляхом успадкування та використання ключового слова "virtual".