НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота №4
з дисципліни “Об’єктно орієнтоване програмування”
тема “C# .Net. Управління ресурсами. Написання чистого коду”

Виконав(ла)	**Перевірив**
студент(ка) II курсу	**“____”“_____”20___р.**
групи КП-83	**викладач**
Коваль Андрій Олекснадрович	**_____**
(прізвище, ім’я, по батькові)	**(прізвище, ім’я, по батькові)**

Київ 2019

Мета роботи

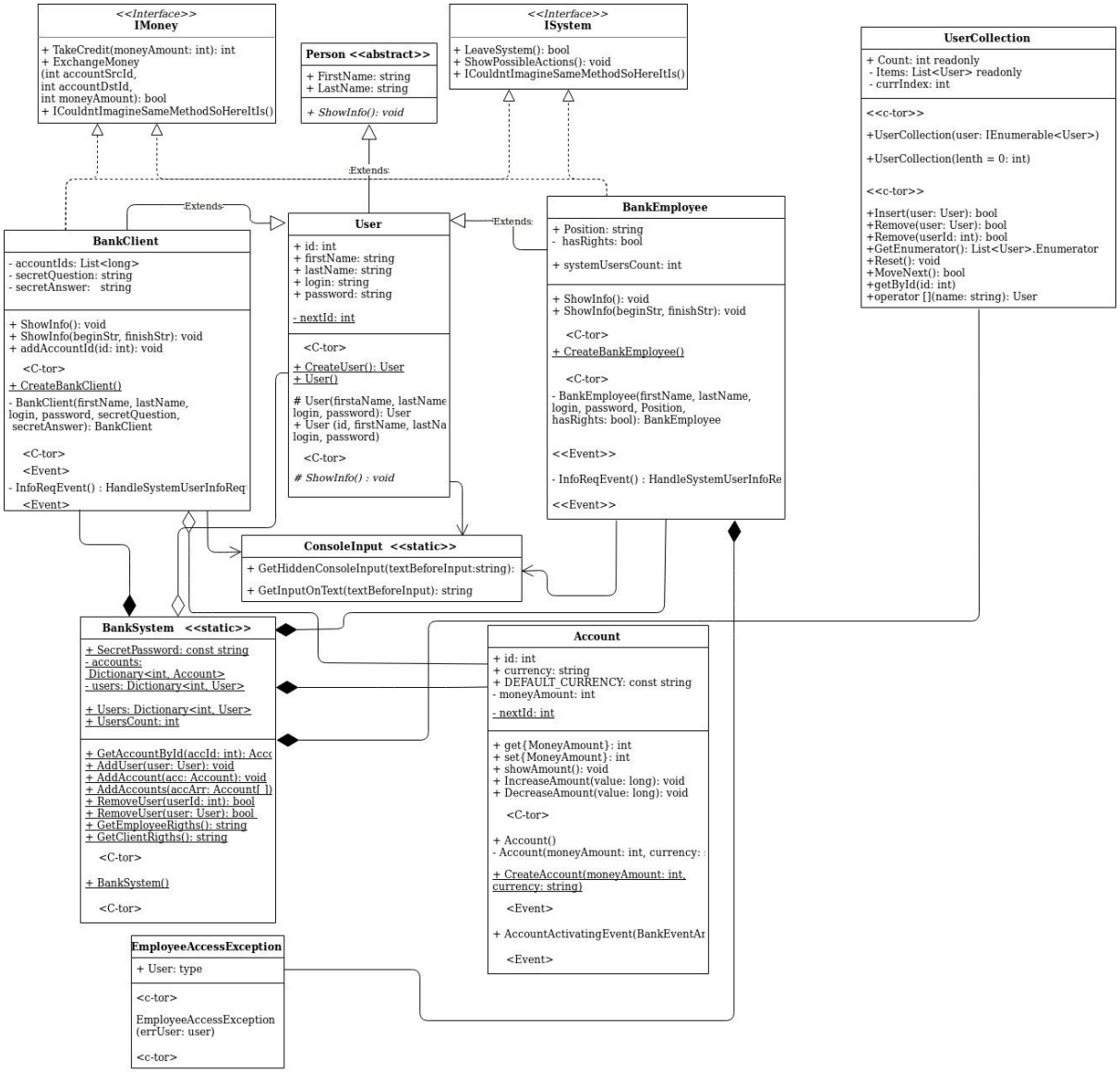
Ознайомитися з правилами написання чистого коду, прийомами рефакторингу, вивчити механізм управління ресурсами, реалізований у .Net.

Постановка завдання

1. Додати до класів, створених в рамках виконання попередніх лабораторних робіт, методи, наявність яких дозволятиме управляти знищенням об’єктів цих класів (3 бали):
    1. реалізувати інтерфейс IDisposable;
    2. створити деструктори;
    3. забезпечити уникнення конфліктів між Dispose та деструктором.
2. 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 (5 балів).
3. Продемонструвати можливості роботи з WeakReference (2 бали).
4. Привести код класів (розроблених у попередніх лабораторних роботах) у відповідність до евристичних правил щодо написання чистого коду (груп “Функції”, “Імена”, “Загальні” з книги Р.Мартіна “Чистий код”), а також до правил рефакторингу. До відповідних фрагментів коду написати коментар із вказівкою правила, якому задовольняє цей фрагмент.
5. Перерахувати всі правила написання чистого коду, яким

задовольняє програмний код. У разі, якщо код не відповідає цим правилам, вказати, який прийом рефакторингу було застосовано для виправлення ситуації. (5 балів)

Діаграма класів



Фрагменти коду програми

IDisposable та деструктор

```
void IDisposable.Dispose()
      {
                  Dispose(true);
            GC.SuppressFinalize(this);
      }
```

```
      private void Dispose(bool disposing)
      {
            //Console.WriteLine($"Disposing method with flag {disposing}
{this.id}");

            if (this._disposed)
                  return;

            if (disposing)
            {
                  // free managed recourses

                  // CLEAN CODE
                  // the best if statements and loops are 1-2 lines
heigth
                  removeAllEventSubscriptions();
            }
            // free unmapped recourses
            // @todo ask

            this._disposed = true;
      }
```

```
// CLEAN CODE - each func MUST do only one thing
      // The best functions are parameterless
      // Each func should cover only one level of abstraction
      private void removeAllEventSubscriptions()
      {
            foreach (Delegate func in
AccountActivatingEvent.GetInvocationList())
            {
                  AccountActivatingEvent -= (AccountHandler)func;
            }
      }
```

```
~Account()
      {
            // CLEAN CODE
            // it's not apropriate to log here but its done to
expliciltly show finalizer
            Console.WriteLine($"d-ctor of acc with id {this.id}");
            Dispose(false);
      }
```

методи Collect, SupressFinalize, ReRegisterForFinalize, GetTotalMemory, GetGeneration, WaitForPendingFinalizers

```
private static void GCDemoBegin()
      {
            Console.WriteLine("Garbage collection demo ---");

            LogTotalMemory("Memory used before allocating");

            int sampleBigNumber = 100;
            Console.WriteLine($"Allocating {sampleBigNumber} accounts");

            var garbageList = makeListOfGarbage(sampleBigNumber);
            var generation = GC.GetGeneration(garbageList[0]);

            LogTotalMemory("Memory used after allocating");

            garbageList.ForEach((acc) => (acc as IDisposable).Dispose());

            GC.Collect(generation);
            LogTotalMemory("Memory after GC.Collect disposed");

            //this makes .net to call d-ctor of finalizer as its
oficially called
            garbageList.ForEach((acc) => GC.ReRegisterForFinalize(acc));
      }
```

```
   private static List<Account> makeListOfGarbage(int count)
      {
            var garbageList = new List<Account>();
            var rand = new Random();
            for (int i = 0; i < count; i++)
            {
                  var acc = new Account(rand.Next(), "uah");
                  garbageList.Add(acc);
            }
            return garbageList;
      }
```

```
private static void LogTotalMemory(string stringAfterMemoryLogged, bool
waitForFullCollectionBeforeLog = true)
      {
            // CLEAN CODE - do not repeat yourself
            stringAfterMemoryLogged =
checkStringAndTrim(stringAfterMemoryLogged);
            var memoryUsed =
GC.GetTotalMemory(waitForFullCollectionBeforeLog);
            string output = $"{memoryUsed}";

            if (stringAfterMemoryLogged.Length != 0)
                  output += $" - {stringAfterMemoryLogged}";

            Console.WriteLine(output);
      }
```

```
private static void GCDemoFinish()
      {
            GC.Collect();
            GC.WaitForPendingFinalizers();
            LogTotalMemory("Exiting stackframe and force GC.Collect
```

```
again");
      }
private static void GCDemo()
      {
            GCDemoBegin();
            GCDemoFinish();
      }
```

WeakReference

```
 public static void weakReferenceDemo()
 {
       var weakAccountRef = new WeakReference(new Account(123123,
"uah"));
       if (weakAccountRef.IsAlive)
       {
           Console.WriteLine("Account ref is instansiated");
       }

       GC.Collect(GC.MaxGeneration, GCCollectionMode.Forced);
       GC.WaitForPendingFinalizers();

       if (weakAccountRef.IsAlive)
       {
                   Console.WriteLine("Account is still alive after
GC.collect");
       }
       else
       {
                   Console.WriteLine($"Account is dead and the value of
weak account reference is {weakAccountRef}");
       }
   }
```

Висновки

Я ознайомився з правилами написання чистого коду, прийомами рефакторингу, вивчив механізм управління ресурсами, реалізований у .Net.