Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)»
Кафедра Информационной безопасности

ОТЧЁТ

по лабораторной работе №8

по дисциплине «ООП»

Тема: «Применение конструкторов C#»

Студент гр. 3363 _____ Минко Д.А. Овсейчик Н.И. Гончаренко О.Д.

Преподаватель _____ Новикова Н.Е.

Санкт-Петербург
2024

Цель работы

Освоение механизмов работы конструкторов и деструкторов в языке программирования C#.

Используемые аппаратные и программные средства

– Лабораторная работа выполнена на операционной системе Arch Linux;

– В качестве среды разработки (IDE) используется NeoVim;

– Используемый компилятор — dotnet;

– Версия SDK — 8.0;

– Отчет написан на LaTex.

ХОД РАБОТЫ

В качестве исходных данных для выполнения лабораторной работы выступает программный код, описывающий банковское приложеные, осуществляющее транзакции между аккаунтами пользователей. В исходном коде описаны следующие программные сущности:

1) Класс CreateAccount, содержащий точку входа программы и предназначенный для инициализации пользовательских аккаунтов и непосредственно выполнения денежных операций с последующим выводом информации об осуществленных транзакциях и информации об аккаунтах;

2) Класс BankAccount описывает пользовательский аккаунт и логику осуществления банковских операций. Класс содержит конструкторы, позволяющие инициализировать аккаунты пользователей при разных начальных данных (например по балансу, по типу и тп.), а также геттеры, возвращающие различную информацию об аккаунте;

3) Класс BankTransaction описывает банковскую транзакцию и позволяет оперировать такими данныме о денежных переводах, как дата и время совершения операции и номинальность операции.

Полный исходный код представлен в приложении 1. Листинг программы отражен на следующей иллюстрации:

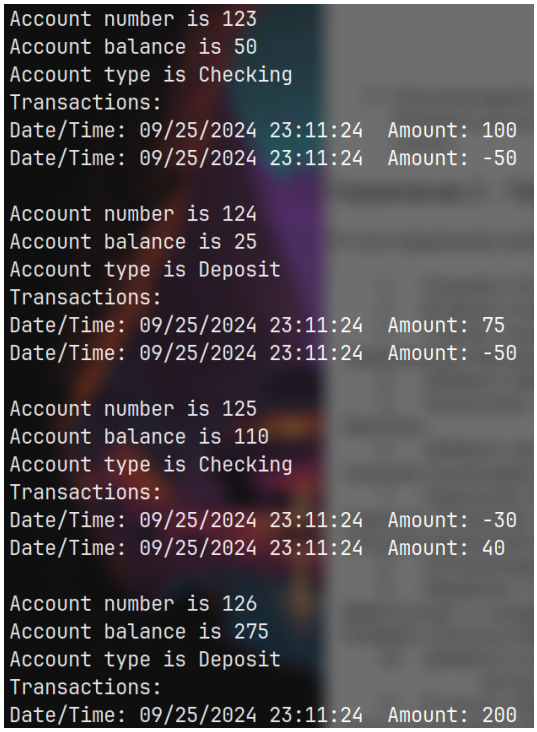

Рисунок 1 — Запуск исходной программы

1.1 Создание деструктора

В ходе выполнения упражнения в класс BankTransaction был добавлен деструктор (финализатор), создающий/открывающий файл Transactions.Dat и записывающий в неого дату и время совершения транзакции. Исходный код деструктора:

```
1 ~BankTransaction()
2 {
3     StreamWriter swFile = File.AppendText("Transactions.Dat");
4     swFile.WriteLine("Date/Time: {0}\tAmount: {1}", when, amount);
5     swFile.Close();
6     GC.SuppressFinalize(this);
7 }
```

Стоит учитывать особенности работы сборщика мусора в C# начиная с версии .Net 5.0 из-за которых при завершении программы деструкторы не вызываются. Освобождение памяти происходит позже, когда система сочтет это необходи-

мым. Таким образом для вызова деструкторов необходимо явно вызвать сборку мусора, но этого будет мало для текущей реализации.

Так как в ней в классе BankAccount используется Queue, сборщик мусора не будет освобождать память экземпляров этого класса до тех пор, пока очередь содержит действительные объекты, поэтому Queue также необходимо освободить вручную.

Для ручного освобождения очереди в классе BankAccount был описан метод Dispose (паттерн в C#, предназначенный для ручного освобождения памяти; «ручной деструктор»). После вызова метода Dispose() экземпляр класса является недействительным (очищенным).

Реализация метода Dispose:

```
class BankAccount : IDisposable
{
    ...
    public void Dispose()
    {
        tranQueue.Clear();
    }
}
```

Ручной вызов сборщика мусора в методе Main:

```
class CreateAccount
{
    static void Main()
    {
        BankAccount acc1;

        acc1 = new BankAccount();
        acc1.Deposit(200);
        acc1.Withdraw(100);
        Write(acc1);

        acc1.Dispose();
```

```
13
14          GC.Collect();
15          GC.WaitForPendingFinalizers();
16      }
17      ...
18  }
```

Полный исходный код программы представлен в приложении 2. В результате работы программы в директории проекта будет создан файл Transactions.Dat со следующим содержимым:

```
Date/Time: 09/26/2024 16:10:31 Amount: -100
Date/Time: 09/26/2024 16:10:31 Amount: 200
```

В исходной реализации программы экземпляры транзакций хранятся в очереди, которая является полем класса BankAccount. По этой причине деструкторы (финализаторы) экземпляров транзакции не вызываются до тех пор, пока они не были выведены из очереди. Так как экземпляры транзакций должны храниться в очереди в течении всего времени существования объекта класса BankAccount для корректного вывода информации о транзакциях, деструкторы транзакций будут выполнены при уничтожении объекта класса BankAccount, из-за чего время транзакций, выведенное в файл, будет соответствовать времени уничтожения экземпляра BankAccount, а не времени совершения транзакции. Таким образом, с точки зрения выполнения поставленных задач данная реализация некорректна.

Для исправления описанного бага содержимое деструктора BankTransaction должно быть перенесено в его конструктор.

ВЫВОД

В результате выполнения лабораторной работы были закреплены на практике теоретические знания о конструкторах и деструкторах в языке программирования C#.

Продуктом лабораторной работы является реализованая программа, описывающая банковское приложение, осуществляющее транзакции между аккаунтами пользователей.

В ходе выполнения лабораторной работы были изучены особенности сборки мусора в современных версиях .Net (начиная с 5.0), которые заключаются в отсутствии автоматического освобождения памяти (в том числе вызовов деструкторов) сборщиком мусора по завершении выполнения программы. Освобождение памяти происходит позже, когда система сочтет это необходимым. Таким образом для ручного освобождения памяти в программе были использованы явный вызов сборщика мусора и патерн Dispose.

Также была выявлена особенность освобождения памяти для экземпляров класса Queue. Память выделенная для них автоматически освобождается только тогда, когда очередь не содержит действительных объектов.

ПРИЛОЖЕНИЕ 1

Файл CreateAccount.cs:

```
class CreateAccount
{
    // Test Harness
    static void Main()
    {
        BankAccount acc1, acc2, acc3, acc4;

        acc1 = new BankAccount();
        acc2 = new BankAccount(AccountType.Deposit);
        acc3 = new BankAccount(100);
        acc4 = new BankAccount(AccountType.Deposit, 500);

        acc1.Deposit(100);
        acc1.Withdraw(50);
        acc2.Deposit(75);
        acc2.Withdraw(50);
        acc3.Withdraw(30);
        acc3.Deposit(40);
        acc4.Deposit(200);
        acc4.Withdraw(450);
        acc4.Deposit(25);

        Write(acc1);
        Write(acc2);
        Write(acc3);
        Write(acc4);
    }

    static void Write(BankAccount acc)
    {
        Console.WriteLine("Account number is {0}",
                acc.Number());
        Console.WriteLine("Account balance is {0}",
```

```
                acc.Balance());
        Console.WriteLine("Account type is {0}",
                acc.Type());
        Console.WriteLine("Transactions:");
        foreach (BankTransaction tran in
                acc.Transactions())
        {
            Console.WriteLine("Date/Time: {0}\tAmount: " +
                    "{1}", tran.When(), tran.Amount());
        }
        Console.WriteLine();
    }
}
```

Файл BankAccount.cs:

```
using System.Collections;

class BankAccount
{
    private long accNo;
    private decimal accBal;
    private AccountType accType;
    private Queue tranQueue = new Queue();

    private static long nextNumber = 123;

    // Constructors
    public BankAccount()
    {
        accNo = NextNumber();
        accType = AccountType.Checking;
        accBal = 0;
    }

    public BankAccount(AccountType aType)
    {
```

```
22          accNo = NextNumber();
23          accType = aType;
24          accBal = 0;
25      }
26
27      public BankAccount(decimal aBal)
28      {
29          accNo = NextNumber();
30          accType = AccountType.Checking;
31          accBal = aBal;
32      }
33
34      public BankAccount(AccountType aType, decimal aBal)
35      {
36          accNo = NextNumber();
37          accType = aType;
38          accBal = aBal;
39      }
40
41      public bool Withdraw(decimal amount)
42      {
43          bool sufficientFunds = accBal >= amount;
44          if (sufficientFunds) {
45              accBal -= amount;
46              BankTransaction tran = new BankTransaction(
47                      -amount);
48              tranQueue.Enqueue(tran);
49          }
50          return sufficientFunds;
51      }
52
53      public decimal Deposit(decimal amount)
54      {
55          accBal += amount;
56          BankTransaction tran = new BankTransaction(amount);
57          tranQueue.Enqueue(tran);
```

```
58          return accBal;
59      }
60
61      public Queue Transactions()
62      {
63          return tranQueue;
64      }
65
66      public long Number()
67      {
68          return accNo;
69      }
70
71      public decimal Balance()
72      {
73          return accBal;
74      }
75
76      public string Type()
77      {
78          return accType.ToString();
79      }
80
81      private static long NextNumber()
82      {
83          return nextNumber++;
84      }
85  }
```

Файл BankTransaction.cs:

```
1  /// <summary>
2  ///   A BankTransaction is created every time a deposit or withdrawal
       occurs on a BankAccount
3  ///   A BankTransaction records the amount of money involved, together
       with the current date and time.
4  /// </summary>
```

```
public class BankTransaction
{
    private readonly decimal amount;
    private readonly DateTime when;

    public BankTransaction(decimal tranAmount)
    {
       amount = tranAmount;
       when = DateTime.Now;
    }

    public decimal Amount()
    {
        return amount;
    }

    public DateTime When()
    {
        return when;
    }
}
```

Файл AccountType.cs:

```
\begin{verbatim}
enum AccountType
{
    Checking,
    Deposit
}
```

ПРИЛОЖЕНИЕ 2

Файл CreateAccount.cs:

```
class CreateAccount
{
    static void Main()
    {
        BankAccount acc1;

        acc1 = new BankAccount();
        acc1.Deposit(200);
        acc1.Withdraw(100);
        Write(acc1);

        acc1.Dispose();
        GC.Collect();
        GC.WaitForPendingFinalizers();
    }

    static void Write(BankAccount acc)
    {
        Console.WriteLine("Account number is {0}",
                acc.Number());
        Console.WriteLine("Account balance is {0}",
                acc.Balance());
        Console.WriteLine("Account type is {0}",
                acc.Type());
        Console.WriteLine("Transactions:");
        foreach (BankTransaction tran in
                acc.Transactions())
        {
            Console.WriteLine("Date/Time: {0}\tAmount: " +
                    "{1}", tran.When(), tran.Amount());
        }
        Console.WriteLine();
    }
```

```
}
```

Файл BankAccount.cs:

```
using System.Collections;

class BankAccount : IDisposable
{
    private long accNo;
    private decimal accBal;
    private AccountType accType;
    private Queue tranQueue = new Queue();

    private static long nextNumber = 123;

    // Constructors
    public BankAccount()
    {
        accNo = NextNumber();
        accType = AccountType.Checking;
        accBal = 0;
    }

    public BankAccount(AccountType aType)
    {
        accNo = NextNumber();
        accType = aType;
        accBal = 0;
    }

    public BankAccount(decimal aBal)
    {
        accNo = NextNumber();
        accType = AccountType.Checking;
        accBal = aBal;
    }

```

```
34      public BankAccount(AccountType aType, decimal aBal)
35      {
36          accNo = NextNumber();
37          accType = aType;
38          accBal = aBal;
39      }
40
41      public bool Withdraw(decimal amount)
42      {
43          bool sufficientFunds = accBal >= amount;
44          if (sufficientFunds) {
45              accBal -= amount;
46              BankTransaction tran = new BankTransaction(
47                      -amount);
48              tranQueue.Enqueue(tran);
49          }
50          return sufficientFunds;
51      }
52
53      public decimal Deposit(decimal amount)
54      {
55          accBal += amount;
56          BankTransaction tran = new BankTransaction(
57                      amount);
58          tranQueue.Enqueue(tran);
59          return accBal;
60      }
61
62      public Queue Transactions()
63      {
64          return tranQueue;
65      }
66
67      public long Number()
68      {
69          return accNo;
```

```
70      }
71
72      public decimal Balance()
73      {
74          return accBal;
75      }
76
77      public string Type()
78      {
79          return accType.ToString();
80      }
81
82      private static long NextNumber()
83      {
84          return nextNumber++;
85      }
86
87      public void Dispose()
88      {
89          tranQueue.Clear();
90      }
91  }
```

Файл BankTransaction.cs:

```
1   using System.IO;
2   /// <summary>
3   ///   A BankTransaction is created every time a deposit
4   ///   or withdrawal occurs on a BankAccount
5   ///   A BankTransaction records the amount of money
6   ///   involved, together with the current date and time.
7   /// </summary>
8   public class BankTransaction
9   {
10      private readonly decimal amount;
11      private readonly DateTime when;
12
```

```
13      public BankTransaction(decimal tranAmount)
14      {
15          Console.WriteLine("Tran created");
16          amount = tranAmount;
17          when = DateTime.Now;
18      }
19
20      public decimal Amount()
21      {
22          return amount;
23      }
24
25      public DateTime When()
26      {
27          return when;
28      }
29
30      ~BankTransaction()
31      {
32          StreamWriter swFile = File.AppendText("Transactions.Dat");
33          swFile.WriteLine("Date/Time: {0}\tAmount: {1}",
34                           when, amount);
35          swFile.Close();
36          GC.SuppressFinalize(this);
37      }
38  }
```

Файл AccountType.cs:

```
1  enum AccountType
2  {
3      Checking,
4      Deposit
5  }
```