

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#»**

Студенты гр. 3363

Минко Д.А. Овсейчик Н.И.  
Гончаренко О.Д.

Преподаватель

Новакова 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 приложение, 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 данными 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:

```
Account number is 123
Account balance is 50
Account type is Checking
Transactions:
Date/Time: 09/25/2024 23:11:24 Amount: 100
Date/Time: 09/25/2024 23:11:24 Amount: -50

Account number is 124
Account balance is 25
Account type is Deposit
Transactions:
Date/Time: 09/25/2024 23:11:24 Amount: 75
Date/Time: 09/25/2024 23:11:24 Amount: -50

Account number is 125
Account balance is 110
Account type is Checking
Transactions:
Date/Time: 09/25/2024 23:11:24 Amount: -30
Date/Time: 09/25/2024 23:11:24 Amount: 40

Account number is 126
Account balance is 275
Account type is Deposit
Transactions:
Date/Time: 09/25/2024 23:11:24 Amount: 200
```

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 него 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`:

```
1 class BankAccount : IDisposable
2 {
3     ...
4     public void Dispose()
5     {
6         tranQueue.Clear();
7     }
8 }
```

Ручной вызов сборщика мусора в методе `Main`:

```
1 class CreateAccount
2 {
3     static void Main()
4     {
5         BankAccount acc1;
6
7         acc1 = new BankAccount();
8         acc1.Deposit(200);
9         acc1.Withdraw(100);
10        Write(acc1);
11
12        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 реализованная 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:

```
1  class CreateAccount
2  {
3      // Test Harness
4      static void Main()
5      {
6          BankAccount acc1, acc2, acc3, acc4;
7
8          acc1 = new BankAccount();
9          acc2 = new BankAccount(AccountType.Deposit);
10         acc3 = new BankAccount(100);
11         acc4 = new BankAccount(AccountType.Deposit, 500);
12
13         acc1.Deposit(100);
14         acc1.Withdraw(50);
15         acc2.Deposit(75);
16         acc2.Withdraw(50);
17         acc3.Withdraw(30);
18         acc3.Deposit(40);
19         acc4.Deposit(200);
20         acc4.Withdraw(450);
21         acc4.Deposit(25);
22
23         Write(acc1);
24         Write(acc2);
25         Write(acc3);
26         Write(acc4);
27     }
28
29     static void Write(BankAccount acc)
30     {
31         Console.WriteLine("Account number is {0}",
32                             acc.Number());
33         Console.WriteLine("Account balance is {0}",
```



```

34         acc.Balance());
35     Console.WriteLine("Account type is {0}",
36         acc.Type());
37     Console.WriteLine("Transactions:");
38     foreach (BankTransaction tran in
39         acc.Transactions())
40     {
41         Console.WriteLine("Date/Time: {0}\tAmount: " +
42             "{1}", tran.When(), tran.Amount());
43     }
44     Console.WriteLine();
45 }
46 }

```

Файл BankAccount.cs:

```

1  using System.Collections;
2
3  class BankAccount
4  {
5      private long accNo;
6      private decimal accBal;
7      private AccountType accType;
8      private Queue tranQueue = new Queue();
9
10     private static long nextNumber = 123;
11
12     // Constructors
13     public BankAccount()
14     {
15         accNo = NextNumber();
16         accType = AccountType.Checking;
17         accBal = 0;
18     }
19
20     public BankAccount(AccountType aType)
21     {

```

```

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>

```

```

5  public class BankTransaction
6  {
7      private readonly decimal amount;
8      private readonly DateTime when;
9
10     public BankTransaction(decimal tranAmount)
11     {
12         amount = tranAmount;
13         when = DateTime.Now;
14     }
15
16     public decimal Amount()
17     {
18         return amount;
19     }
20
21     public DateTime When()
22     {
23         return when;
24     }
25 }

```

Файл AccountType.cs:

```

1  \begin{verbatim}
2  enum AccountType
3  {
4      Checking,
5      Deposit
6  }

```

## ПРИЛОЖЕНИЕ 2

Файл CreateAccount.cs:

```
1  class CreateAccount
2  {
3      static void Main()
4      {
5          BankAccount acc1;
6
7          acc1 = new BankAccount();
8          acc1.Deposit(200);
9          acc1.Withdraw(100);
10         Write(acc1);
11
12         acc1.Dispose();
13         GC.Collect();
14         GC.WaitForPendingFinalizers();
15     }
16
17     static void Write(BankAccount acc)
18     {
19         Console.WriteLine("Account number is {0}",
20             acc.Number());
21         Console.WriteLine("Account balance is {0}",
22             acc.Balance());
23         Console.WriteLine("Account type is {0}",
24             acc.Type());
25         Console.WriteLine("Transactions:");
26         foreach (BankTransaction tran in
27             acc.Transactions())
28         {
29             Console.WriteLine("Date/Time: {0}\tAmount: " +
30                 "{1}", tran.When(), tran.Amount());
31         }
32         Console.WriteLine();
33     }
```

34 }

Файл BankAccount.cs:

```
1  using System.Collections;
2
3  class BankAccount : IDisposable
4  {
5      private long accNo;
6      private decimal accBal;
7      private AccountType accType;
8      private Queue tranQueue = new Queue();
9
10     private static long nextNumber = 123;
11
12     // Constructors
13     public BankAccount()
14     {
15         accNo = NextNumber();
16         accType = AccountType.Checking;
17         accBal = 0;
18     }
19
20     public BankAccount(AccountType aType)
21     {
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(
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  }

```