4. Листинг 5.2. Исходный код **BankSimulation.cs.** Программа моделирования работы в банке

```
001: using System; // Программа реализует доступ и управление несколькими счетами
002:
003: namespace ConsoleApplication_ BankSimulation
004: {
005: /*************************************************/
006: class Account  // Account (счет) - это пользовательский класс; Account - это объект
007: {                    // Объект Account должен хранить (т.е. содержать) след-е переменные экземпляра:
008: private decimal balance;                  // 1) свой баланс (то есть баланс данного счета),
009: private decimal currentInterestRate;    // 2) свою текущую процентную ставку
010: private decimal totalInterestPaid;       // 3) сумму начисленных процентов
011:
012: public Account() // Это конст-р Account, т.е. м-д с тем же им-ем, что и класс Account, в кот-м он нах-ся
013: {                        // Назначение конст-ра - инициализация переменных экземляра  класса Account
014:    balance = 0;
015:    currentInterestRate = 0;
016:    totalInterestPaid = 0;
017: }
018:                          //  # 3.1.1
019: public void SetInterestRate(decimal newInterestRate) // М-д (мутатор) – уст-вить проц-ную ставку
020: {
021:    currentInterestRate = newInterestRate;
022: }
023:
024: public decimal  GetInterestRate()     // Метод (аксессор) - получить процентную ставку   # 4.1.2
025: {                                                                                          //    # 7.1.1
026:    return  currentInterestRate;
027: }
028                          // # 4.1.3
029: public void  UpdateInterest()  // Метод (мутатор) : 1) определение суммы начисленных процентов
030: {                                    // 2) определение баланса с учетом суммы начисленных процентов
031:    totalInterestPaid += balance * currentInterestRate;
032:    balance += balance * currentInterestRate;
033: }
034:                                 //   # 6.1.1
```

```
035: public decimal GetTotalInterestPaid()  // М-д (аксессор) - получить значение суммы начис-х про-тов
036: {
037:     return totalInterestPaid;
038:}
039:                                    //  # 1.1.1
040: public void Deposit(decimal amount)  // М-д (мутатор) - определение баланса (положить вклад)
041: {
042:     balance += amount;
043: }
044:                                   //  # 2.1.1
045: public void Withdraw(decimal amount) // М-д (мутатор) - определение баланса (снять со счета)
046: {
047:     balance -= amount;
048: }
049:
050: public decimal GetBalance()                //   # 4.1.1 - М-д (аксессор) - получение значения баланса
051: {                                          //   # 1.1.2   ;   # 2.1.2   ;     //   # 5.1.1
052:     return balance;
053: }
054: }
055: /*******************************************************/
056: class Bank     // кл-с Bank сод-жит колл-цию сч-ов. Все м-ды кл-са пред-ны для изменения или чтения
057: {  // информации одного счета или всей коллекции счетов и, след-но, зависят от мет-ов класса Account
058: private Account [ ] accounts; // кл-су Банк треб-ся только одна перем-я экз-ра – мас-в счетов accounts
059:
060: public Bank() // Констр-р Bank – созд-т новые объекты Account, которыми класс Bank будет управлять
061:  {
062:     Console.WriteLine("Поздравляем! Вы создали новый банк");
063:     Console.Write("Пожалуйста, введите количество счетов в банке: ");
064:     accounts = new Account[Convert.ToInt32(Console.ReadLine())];
065:     for (int i = 0; i < accounts.Length; i++)
066:     {
067:     accounts[i] = new Account();      // присв-е ссылок на объекты Account элементам массива accounts
068:     }
069:  }
070:
```

```
071: public void Deposit() // Положить средства на указанный счет                    # 1.1
072: {
073:     int accountNumber; // индекс мас-ва accounts. Для польз-ля первый инд-с = 1, для прог-мы - нулю.
074:     decimal amount;
075:     Console.Write("Положить средства. Пожалуйста, введите номер счета: ");
076:     accountNumber = Convert.ToInt32(Console.ReadLine());
077:     Console.Write("Введите объем вклада: ");
078:     amount = Convert.ToDecimal(Console.ReadLine());   // amount - объем вклада
079:     accounts[accountNumber - 1].Deposit(amount);  // 040 – 043; мас-в accounts инд-ся с нуля, поэтому
080:     Console.WriteLine("Новый баланс счета {0}: {1:C}",    // из инд-а выч-ся 1 [accountNumber - 1].
081:             accountNumber, accounts[accountNumber - 1].GetBalance());   // 050 – 053
082: }
083:
084: public void Withdraw() //  Снять средства с указанного счета                    # 2.1
085: {
086:     int accountNumber;
087:     decimal amount;
088:     Console.Write("Снять средства. Пожалуйста, введите номер счета: ");
089:     accountNumber = Convert.ToInt32(Console.ReadLine());
090:     Console.Write("Введите объем снимаемых средств: ");
091:     amount = Convert.ToDecimal(Console.ReadLine()); // amount - объем снимаемых средств со счета
092:     accounts[accountNumber - 1].Withdraw(amount);                    //  045 - 048
093:     Console.WriteLine("Новый баланс счета {0}: {1:C}",
094:             accountNumber, accounts[accountNumber - 1].GetBalance());   //  050 - 053
095: }
096:
097: public void SetInterestRate()  //  Установить процентную ставку указанного счета   # 3.1
098: {
099:     int accountNumber;
100:     decimal newInterestRate;
101:     Console.Write("Установите процентную ставку. Пожалуйста, введите номер счета: ");
102:     accountNumber = Convert.ToInt32(Console.ReadLine());
103:     Console.Write("Введите процентную ставку: ");
104:     newInterestRate = Convert.ToDecimal(Console.ReadLine());
105:     accounts[accountNumber - 1].SetInterestRate(newInterestRate);          //  019 - 022
106: }
```

```
107:
108: public void PrintAllInterestRates()   //   Вывести процентную ставку по каждому счету # 7.1
109: {
110:    Console.WriteLine("Процентная ставка для всех счетов: ");
111:    for (int i = 0; i < accounts.Length; i++)
112:    {
113:             Console.WriteLine("Счета {0,-3}: {1,-10}",
114:                      (i+1), accounts[ i ].GetInterestRate());                    //  024 - 027
115:    }
116: }
118:
119: public void PrintAllBalances()       //   Вывести балансы всех счетов  # 5.1
120: {
121:    Console.WriteLine("Баланс счета для всех счетов: ");
122:    for (int i = 0; i < accounts.Length; i++)
123:    {
124:             Console.WriteLine("Счета {0,-3}: {1:C}",
125:                      (i+1), accounts[ i ].GetBalance());              //   050 - 053
126:    }
127: }
128:
129: public void PrintTotalInterestPaidAllAccounts()  // Выв-ти сумму проц-в, начисл-х по кажд. сч-у  # 6.1
130: {
131:    Console.WriteLine("Общая процентная ставка, оплаченная за каждый индивидуальный счет:");
132:    for (int i = 0; i < accounts.Length; i++)
133:    {
134:             Console.WriteLine("Счета {0,-3}: {1:C}",
135:                      (i+1), accounts[ i ].GetTotalInterestPaid());              //   035 - 038
136:    }
137: }
138:
139: public void UpdateInterestAllAccounts()     //  Добавить проценты по всем счетам  # 4.1
140: {
141:    for (int i = 0; i < accounts.Length; i++)
142:    {
143:             Console.WriteLine("Процентная ставка, добавленная к счету номер {0,-3}: {1:C}",
```

```
144:                                    (i+1), accounts[i].GetBalance() * accounts[i].GetInterestRate());
145:            accounts[ i ].UpdateInterest();          // 050 – 053;   024 – 027;   029 - 033
146:    }
147: }
148: }
149: /*****************************************************/
150: class BankSimulation // класс Моделирование Банка
151: {
152: private static Bank bigBucksBank; // кл-су BankSimulation треб-ся одна пер-я экз-ра - объект Bank
153:
154: public static void Main()
155:    {
156:    string command;
157:
158:    bigBucksBank = new Bank(); // Создание нового объекта Bank
159:    do
160:    {
161:            PrintMenu();
162:            command = Console.ReadLine().ToUpper();
163:            switch(command)
164:            {
165:            case "D":
166:                    bigBucksBank.Deposit();                  // 071 – 082 , # 1
167:                    break;
168:            case "W":
169:                    bigBucksBank.Withdraw();                 // 084 - 095,   # 2
170:                    break;
171:            case "S":
172:                    bigBucksBank.SetInterestRate();          // 097 – 106 , # 3
173:                    break;
174:            case "U":
175:                    bigBucksBank.UpdateInterestAllAccounts();  // 139 – 147 , # 4
176:                    break;
177:            case "P":
178:                    bigBucksBank.PrintAllBalances();          // 119 - 127,  # 5
179:                    break;
```

```
180:            case "T":
181:                    bigBucksBank.PrintTotalInterestPaidAllAccounts();  // 129 – 137 , # 6
182:                    break;
183:            case "I":
184:                    bigBucksBank.PrintAllInterestRates();                  // 108 – 116 , # 7
185:                    break;
186:            case "E":
187:                    Console.WriteLine("Bye Bye!");
188:                    break;
189:            default:
190:                    Console.WriteLine("Неправильный выбор");
191:                    break;
192:            }
193:    } while (command != "E");
194:    Console.ReadLine();
195: }
196:
197: private static void PrintMenu()
198: {
199: Console.WriteLine("\nЧто Вы желаете сделать?\n" +
200: "D)eposit - положить средства на указанный счет\n" +
201: "W)ithdraw - снять средства с указанного счета\n" +
202: "S)et interest rate - установить процентную ставку указанного счета\n" +
203: "U)pdate all accounts for interest - добавить проценты ко всем счетам\n" +
204: "P)rint all balances - вывести балансы всех счетов\n" +
205: "T)otal interest paid printed for all accounts – выв-ти сумму проц-тов, нач-х по каж-му счету\n" +
206: "I)nterest rates printed for all accounts - вывести процентную ставку по каждому счету\n" +
207: "E)nd session - завершить моделирование\n" +
208: "Note: First account has account number one - первый счет имеет индекс один");
209: }
210: }
211: }
```

Приложение 2

Диаграмма взаимодействия модулей c#–программы "Моделирование работы банка"