Объектно-ориентированный анализ и программирование.
Учебно-методический комплекс дисциплины

Забудский Евгений Иванович, доктор технических наук, профессор, e-mail: zei@inbox.ru ,
профессор кафедры Основы информатики и прикладного программного обеспечения,
факультет Бизнес-Информатика, ГУ-ВШЭ

Основная цель, которую необходимо достичь в результате обучения дисциплине Объектно-ориентированный анализ и программирование – научить студентов разрабатывать компьютерные модели реальных систем соответствующих направлению Бизнес-информатика[1].

В процессе изучения дисциплины рассматриваются следующие вопросы**:** объектно-ориентированный анализ (ООА), объектно-ориентированное проектирование (ООПр), объектно-ориентированное программирование (ООП), шаблоны проектирования, унифицированный язык моделирования UML (**U**nified **M**odeling **L**anguage), объектно-ориентированный язык программирования C_Sharp (C#) и другие аспекты ООП [1].

В основе всех этих вопросов лежит один и тот же фундамент**:** *способность и необходимость мыслить категориями объектов реального мира*, так как специалисту-программисту необходимо разрабатывать Windows-приложения, эмулирующие те или иные *системы реального мира*. Поэтому изучение концепции объектного подхода не заканчивается изучением отдельно взятого метода или набора средств разработки. Иными словами, объектный подход является образом *объектно-ориентированного мышления*, которому также необходимо обучить студентов.

Важно понимать, что между изучением концепции объектов и использованием основанных на ней методов и средств существует значительная разница. Необходимо искать такие варианты обучения и обучающие материалы, в которых навыки мышления категориями объектов ставятся выше, чем навыки использования основанных на ней методов и средств.

Таким образом, осваивая объектно-ориентированные методы разработки приложений, важно овладевать фундаментальными принципами объектно-ориентированного подхода и уделять внимание освоению объектно-ориентированного мышления. Поэтому не только на первых лекциях и практических занятиях, но и на всех последующих этому уделяется первостепенное значение.

Переходить на новый способ мышления всегда непросто, поэтому вербальный метод обучения сопровождается активным привлечением компьютерных и информационных технологий. Это позволяет сопровождать рассуждения о концепциях объектов демонстрацией и анализом соответствующих фрагментов программного кода, а также иллюстративной графики.

На лекционных занятиях лектор пользуется компьютером, включенным в Internet. Практические занятия проводятся в компьютерных классах. На всех видах занятий используется мультимедийный проектор. Основными средами, в которых работают и преподаватель и студенты являются: Microsoft Visual Studio **.**NET 2005; Rational Rose; World Wide Web.

Используются следующие Internet-ресурсы.

- ✓ Новые книги раздела C# – http://books.dore.ru/bs/f6sid16.html .
- ✓ *C#* и *.NET* по шагам – http://www.firststeps.ru .
- ✓ *UML* – язык графического моделирования – http://www.uml.org .
- ✓ *JUnit* – каркас тестирования для испытания *Java*-классов – http://www.junit.org .
- ✓ Пакет объектного моделирования *Rational Rose* – http://www-306.ibm.com/software/rational .

[1] Упомянуто данное направление, так как курс читается студентам факультета Бизнес-Информатика ГУ-ВШЭ

Сложность обучения дисциплине также заключается в *дефиците выделенного аудиторного времени*: 16 лекций и 16 практических занятий. Поэтому *особое внимание уделяется организации самостоятельной работы студентов и ее методическому обеспечению.*

С этой целью автором доклада разработан Учебно-методический комплекс дисциплины «Объектно-ориентированный анализ и программирование», который помещен на сайте ГУ-ВШЭ [2]. Комплекс включает: методические рекомендации преподавателям и студентам, программу дисциплины и нормативно-справочные материалы, материалы к лекциям, материалы к практическим занятиям, темы и рекомендации по выполнению курсовой работы и др. Все материалы комплекса доступны студентам и используются ими в процессе обучения.

Для иллюстрации приводится перечень материалов к практическим занятиям и соответствующие гиперссылки, основные вопросы, рассматриваемые на занятии, и объем файлов.

# практического занятия	Тема занятия	Объем файла, КБ
Занятие 0	Введение в **.NET**	235
Занятие 1	Парадигмы процедурно- и объектно-ориентированного программирования. Анализ элементов C#-программ	474
Занятие 2	Демонстрация элементов языка C#	202
Занятие 2. Доп.1	Конструкторы, Свойства	214
Занятие 2. Доп.2	Индексаторы	214
Занятие 3	Абстракция, Инкапсуляция. **Моделирование работы лифта**, UML-диаграммы взаимодействия классов	986
Занятие 4	Типы C# . Форматирование числовых значений. Метаданные. Компонентно-ориентированное программирование	980
Занятие 5	**Моделирование работы банка** и системы лифтов здания. Программа Моделирование работы банка , диаграмма взаимодействия модулей	606 222
Занятие 6	Наследование. Часть I: Базовый и производный классы. Переопределение и перегрузка методов	738
Занятие 7	Наследование. Часть II: Абстрактные функции. Полиморфизм. Интерфейсы	812
Занятие 8	Разработка Windows-приложения (формат **GUI**) “Проект **Расчет оценки студента**”	1151
Занятие 9	Графический интерфейс пользователя (**GUI**). Разработка проекта **NetCalc** (Калькулятор) с использованием встроенного дизайнера форм – средства автоматизации разработки **VS .NET**	1761

В материалах к практическим занятиям и лекциям приведены**:** тема и примерное количество аудиторных часов, отводимых на ее изучение; содержание; теория и листинги программ; резюме; контрольные вопросы и задания (упражнения) по объектно-ориентированному программированию; список литературы; ссылки на Internet-ресурсы и др.

Ниже приводится Содержание **5**-го практического занятия, темой которого является «**Моделирование работы банка и системы лифтов здания**».

Содержание

1. Массивы и классы 4
1.1. Элементы массива как ссылки на объекты 4
1.2. Листинг 5.1. Для эффективного управления коллекцией *лифтов* используется массив объектов – исходный код ElevatorArray.cs **..... 5**
1.3. Массивы как переменные экземпляра в классах 7
1.4. Программа моделирования работы банка 8
1.5. Спецификации программы 8
2. Разработка программы моделирования работы банка 8
2.1. Разделение каждой подсистемы на объекты (модули) 8
2.2. Идентификация переменных экземпляров классов 8
2.3. Идентификация методов в каждом модуле 9
2.3.1. Класс Account 9
2.3.2. Класс Bank 9
2.3.3. Класс BankSimulation 9
3. Разработка внутренних методов 10
4. Листинг 5.2. Проект **Моделирование работы *банка*.** Исходный код BankSimulation.cs **..... 10**
4.1. Результаты работы программы BankSimulation.cs 14
5. Двумерный массив 15
5.1. Объявление и определение двумерного массива 16
5.2. Доступ к элементам двумерного массива **(Листинги 5.3 и 5.4)..... 17**
5.3. Листинг 5.5. Отслеживания вызовов *лифта* в течение каждого часа за семидневный срок – исходный код ElevatorRequestTracker.cs **..... 18**
5.4. Результаты работы программы ElevatorRequestTracker.cs 19
5.5. Свободные ("зубчатыми") массивы 20
5.6. Листинг 5.6. Сбор количества вызовов *лифта* каждый день за различное число часов – исходный код JaggedElevatorRequests.cs **..... 21**
5.7. Результаты работы программы – JaggedElevatorRequests.cs 22
6. Анатомия класса 23
6.1. Анатомия класса: обзор 23
6.2. Переменные-члены 25
6.2.1. Переменные экземпляра 25
6.2.2. Переменные static 26
6.3. Листинг 5.7. Открытие и закрытие *банковских* счетов в течение времени исполнения программы – исходный код DynamicBankSimulation.cs **28**
6.4. Результаты работы программы – DynamicBankSimulation.cs **..... 30**
Контрольные вопросы **33**
Упражнения по программированию **34**
Список литературы 35
П р и л о ж е н и е. **C#** и **.NET** по шагам:
http://www.firststeps.ru/dotnet/dotnet1html 36

На практических занятиях проводится блиц-опрос студентов в соответствии с контрольными вопросами (**с. 33**) и осуществляется проверка выполнения упражнений по программированию (**с. 34**).

Далее, на практических занятиях студентам кратко разъясняются основные положения, раскрывающие тему; затем студенты реализуют в среде Visual Studio **.**NET соответствующие программные продукты (**листинги 5.1 … 5.7**), представленные в учебных материалах, и выполняют их анализ. Так как студенты не успевают ввести и проанализировать все программы в аудитории, то часть из них студентам поручается проработать во *вне*аудиторное время.

Предусмотрены письменные домашняя работа и аудиторная контрольная работа. Письменные работы включают разработку, кодирование, тестирование и отладку объектно-ориентированных программ, реализующих решение нескольких задач в соответствии с изучаемой темой.

Запланировано выполнение курсовой работы. Ее содержанием является компьютерное моделирование реальных или концептуальных систем средствами среды Rational Rose, среды Visual Studio **.**NET и языка C_Sharp в соответствии с парадигмой *компонентно-ориентированного* программирования. С целью оказания помощи студентам в выполнении курсовой работы на практических занятиях осуществляется компьютерное моделирование двух типических реальных систем (см. Материалы к Практическим занятиям # # 3, 5, 8). Далее приводится исходный код программы Моделирование работы банка (**листинг 5.2**) [2, 3].

Листинг 5.2. Программа моделирования работы банка (файл **BankSimulation.cs**)

```
001: using System; // Программа реализует доступ и управление несколькими счетами
002:
003: namespace ConsoleApplication_ BankSimulation
004: {
005: ////////////////////////////////////////////////// начало класса Account //////////////////////////////////////////////////

006: class Account  // Account  – это пользовательский класс
007: {              // Объект Account (счет) должен хранить следующие переменные экземпляра:
008: private decimal balance;                    // 1) свой баланс,
009: private decimal currentInterestRate;       // 2) свою текущую процентную ставку,
010: private decimal totalInterestPaid;         // 3) сумму начисленных процентов
011:
012: public Account() // Инициализация переменных экземляра  класса Account
013: {
014:     balance = 0;
015:     currentInterestRate = 0;
016:     totalInterestPaid = 0;
017: }
018:                                                                    //  # 3.1.1
019: public void SetInterestRate(decimal newInterestRate) //  Установить процентную ставку
020: {
021:     currentInterestRate = newInterestRate;
022: }
023:
024: public decimal  GetInterestRate()     // Получить процентную ставку   # 4.1.2
025: {                                                                   //  # 7.1.1
026:     return  currentInterestRate;
027: }
028                                                                     // # 4.1.3
029: public void  UpdateInterest()    // 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() // Получить значение суммы начислен-х процентов
036: {
037:     return totalInterestPaid;
038:}
039:                                                                    //  # 1.1.1
040: public void Deposit(decimal amount)  // Определение баланса (положить деньги на счет)
041: {
042:     balance += amount;
043: }
044:                                                                    //  # 2.1.1
045: public void Withdraw(decimal amount) // Определение баланса (снять деньги со счета)
046: {
047:     balance -= amount;
048: }
049:
050: public decimal GetBalance()              //  Получение значения баланса – # 1.1.2
051: {                                              //   # 2.1.2   ;   # 4.1.1  ;   //   # 5.1.1
052:     return balance;
053: }
054: }
055: ////////////////////////////////////////////////// конец класса Account ////////////////////////////////////////////

056: class Bank     // Класс Bank содержит коллекцию счетов
057: {                // Счета обрабатываются методами класса Account
058: private Account [ ] accounts; // Переменная экземпляра – массив счетов accounts
059:
060: public Bank()
061:  {    // Создание объектов Account (счетов), которыми класс Bank будет управлять
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();
068:     }
069:  }
070:
071: public void Deposit()                          // Положить деньги на счет –  # 1.1
072: { // accountNumber – индекс массива счетов. Для пользователя первый индекс равен 1,
073:     int accountNumber;                                          // а для программы - нулю
074:     decimal amount;
075:     Console.Write("Положить деньги. Пожалуйста, введите номер счета: ");
076:     accountNumber = Convert.ToInt32(Console.ReadLine());
077:     Console.Write("Введите объем вклада: ");
078:     amount = Convert.ToDecimal(Console.ReadLine());
079:     accounts[accountNumber - 1].Deposit(amount);                    // 040 – 043
080:     Console.WriteLine("Новый баланс счета {0}: {1:C}",
081:             accountNumber, accounts[accountNumber - 1].GetBalance());   // 050 – 053
082: }
083:
084: public void Withdraw()                        //  Снять средства со 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());
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 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:                            // Вывести процентную ставку по каждому счету – # 7.1
108: public void PrintAllInterestRates()
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:            // Вывести сумму процентов, начисленных по каждому счету – # 6.1
129: public void PrintTotalInterestPaidAllAccounts()
130: {
131:    Console.WriteLine("Общая процентная ставка, оплаченная за каждый 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: ////////////////////////////////////////////// конец класса Bank //////////////////////////////////////////////
150: class BankSimulation // класс Моделирование Банка
151: {
152: private static Bank bigBucksBank; // Переменная экземпляра - объект класса Bank
153:
154: public static void Main()
155:     {
156:     string command;
157:
158:     bigBucksBank = new Bank();      // Создание нового объекта класса Bank
159:     do
160:     {
161:             PrintMenu();
162:             command = Console.ReadLine().ToUpper();
163:             switch(command)          // Ниже указаны: 1) номера строк, выполняемых при
164:             {           // нажатии на клавишу с буквой  "D", "W",…,"I" или "E"; 2) номера
165:             case "D":  //  1, 2, …, 8 соответствующие пунктам меню (строки 200-208)
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!");                                  // # 8
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" +          // Пункты меню:
200: "D) – Положить деньги на счет\n" +                                  // # 1
```

```
201: "W) – Снять средства со счета\n" +                                    // # 2
202: "S) – Установить процентную ставку счета\n" +                         // # 3
203: "U) – Добавить проценты ко всем счетам\n" +                           // # 4
204: "P) – Вывести балансы всех счетов\n" +                                // # 5
205: "T) – Вывести сумму процентов, начисленных по каждому счету\n" +      // # 6
206: "I) – Вывести процентную ставку по каждому счету\n" +                 // # 7
207: "E) – Завершить моделирование\n" +                                    // # 8
208: "Примечание: первому счету соответствует индекс равный единице");
209: }
210: }////////////////////////////////////////// конец класса BankSimulation //////////////////////////////////////////
211: }
```

Программа имеет 211 строк, то есть является достаточно большой для рассмотрения на практическом занятии. Для удобства выполнения анализа строки пронумерованы (номера строк и знак «**:**» в программный код не входят). Кроме того, для удобства объяснения составлена Диаграмма взаимодействия модулей C#-программы «Моделирование работы банка» (см. след. страницу). На диаграмме указаны номера строк и приведены номера возле стрелок, которые также проставлены в тексте программы. Эти номера, и комментарии к ним, выделены в тексте программы заливкой серого цвета. Нумерация, принятая в диаграмме, и соответствующая нумерация в тексте программы позволяют преподавателю с минимальной затратой времени разъяснить логику работы C#-программы «Моделирование работы банка».

Покажем это на примере выполнения одной банковской операции «***Положить деньги на счет***», то есть на примере реализации одного пункта меню #**1** (строка **200** кода):

1) предполагаем, что это меню уже отображено на консоли (строки **200 - 208**);

2) после последовательного нажатия на клавишу с буквой “**D**”и клавишу “Enter” будет выполнена **166**-я строка программного кода (см. также **5**-ю строку на *диаграмме* класса BankSimulation);

3) в соответствии с этим будет послано сообщение-запрос методу Deposit(), код которого расположен в строках **071 – 082**.

В строке **071** приведен комментарий // ***Положить деньги на счет –*** **# 1.1**, где первая цифра “**1**” соответствует пункту меню #**1** («Положить деньги на счет»), а вторая цифра “**1**” – **первому** сообщению, направленному методу Deposit() (см. также **5**-ю строку на *диаграмме* класса Bank);

4) метод Deposit() выполняет запрос, и клиент банка вводит значение суммы денег, которые он хочет положить на свой счет. Далее посылается сообщение (из строки 079) специальному методу-мутатору Deposit(**decimal** amount), код которого расположен в строках **040 – 043**. В соответствии с этим методом устанавливается новое значение баланса счета обслуживаемого клиента.

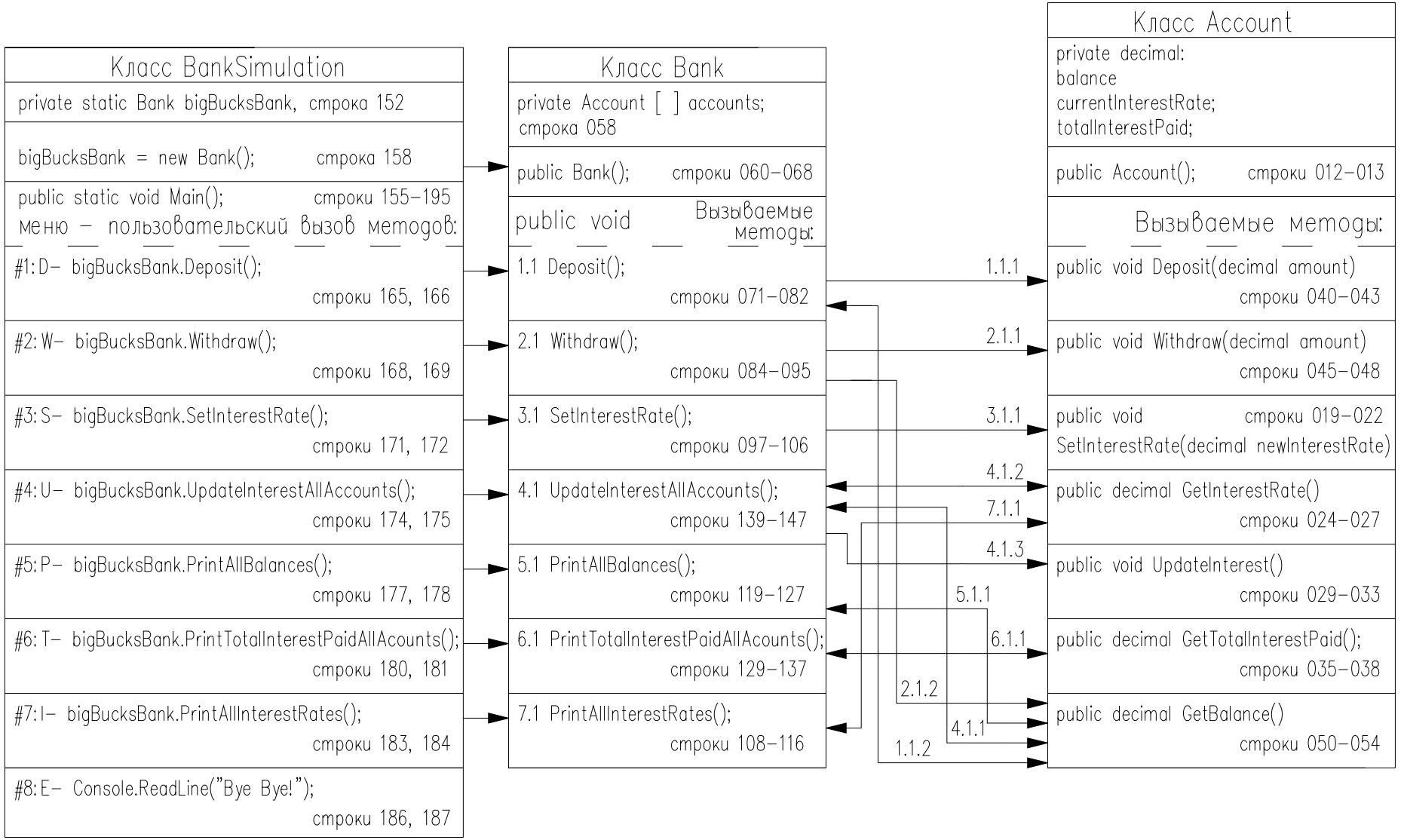
В строке **040** приведен комментарий // ***Определение баланса (положить деньги на счет)*** и выше указан номер **// # 1.1.1**, где смысл первых двух цифр пояснен в пункте *3)*. Третья цифра “**1**” соответствует **первому** сообщению из метода Deposit() – оно направлено методу-мутатору Deposit(**decimal** amount) (см. также **5**-ю строку на *диаграмме* класса Account);

5) после вычисления значения баланса продолжается обслуживание клиента, которое реализовано в методе Deposit(). Посылается сообщение (из строки 081) специальному методу-аксессору GetBalance(), код которого расположен в строках **050 – 053**. В соответствии с этим методом новое значение баланса счета сообщается обслуживаемому клиенту.

В строке **050** приведен комментарий // ***Получение значения баланса*** **– # 1.1.2**, где смысл первых двух цифр пояснен в пункте *3)*, а третья цифра “**2**” соответствует **второму** сообщению из метода Deposit() – оно направлено методу-аксессору GetBalance() (см. также **11**-ю строку на диаграмме класса Account).

На этом банковская операция «***Положить деньги на счет***» завершена. Анализируя пункты *3)* и *4)* соответственно нетрудно сообразить, почему на диаграмме линия с номером 1.1.1 имеет одну стрелку, а линия с номером 1.1.2 – две стрелки.

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



Совершенно аналогично рассмотренным пунктам *1)…5)* выполняются другие банковские операции (см. программу и диаграмму).

Далее приводятся **Результаты работы программы** (*выделено курсивом*), при моделировании выполнения одной банковской операции «***Положить деньги на счет***», рассмотренной выше.

Поздравляем! Вы создали новый банк
Пожалуйста, введите количество счетов в банке: **5** <Enter>

Что Вы желаете сделать?

D) – *Положить средства на указанный счет*
W) – *Снять средства с указанного счета*
S) – *Установить процентную ставку указанного счета*
U) – *Добавить проценты ко всем счетам*
P) – *Вывести балансы всех счетов*
T) – *Вывести сумму процентов, начисленных по каждому счету*
I) – *Вывести процентную ставку по каждому счету*
E) – *Завершить моделирование*
Примечание: первому счету соответствует индекс равный единице

D <Enter>

Положить средства. Пожалуйста, введите номер счета: **2** <Enter>

Введите объем вклада: **10000** <Enter>

*Новый баланс счета **2**: **10 000,00 р**.*
……

Подобные, относительно большие моделирующие программы, рассматриваются только на практических занятиях, а на лекциях анализируются небольшие фрагменты кода, иллюстрирующие теоретические положения парадигмы объектно-ориентированного программирования и способствующие развитию *объектно-ориентированного мышления* у студентов.

На сайте ГУ-ВШЭ, наряду с темами курсовых работ (студенты могут предлагать свои темы) и рекомендациями по их выполнению, также опубликованы *основные* этапы компьютерного моделирования реальных и концептуальных систем с указанием тех страниц разработанных **учебных материалов**, на которых рассмотрены эти этапы. Далее приводится их перечень.

1. Сформулировать целевую функцию моделирования (например, собрать статистическую информацию о качестве функционирования системы >> см. материалы к практическому занятию (ПЗ) **№3 с. 11**).

2. Выполнить анализ реальной системы: учесть её важные действия и атрибуты и отбросить вторичные (реализация концепции ООА – абстракция >> см. материалы к ПЗ **№3, с. 4**).

3. Выделить классы объектов, входящих в реальную систему >> см. материалы к ПЗ **№1, с.9**.

4. Установить связь между классами; выстроить иерархию классов и проанализировать целесообразность повторного использования их кода (реализация концепции ООП - наследование >> см. материалы к ПЗ **№6 с. 4**)

5. Определить объекты классов – экземпляры классов >> см. материалы к ПЗ **№1, с. 2**.

6. Сформулировать переменные экземпляров, которые отражают атрибуты объектов.

7. Установить какие действия должны выполнять объекты, то есть определить методы объектов и их функциональность.

8. Обдумать степень открытости методов и переменных экземпляров (реализация концепции ООП – инкапсуляция >> см. материалы к ПЗ **№3 с. 4**).

9. Выполнить внутреннее проектирование методов; обдумать целесообразность обращения к различным реализациям методов одним и тем же вызовом (реализация концепции ООП – полиморфизм > механизм динамического связывания >> см. материалы к ПЗ **№7 с. 8**).

10. Приступить к разработке исходного кода. На заключительном этапе реализовать парадигму компонентно-ориентированного программирования >> см. материалы к ПЗ **№4 с. 21.**

11. В процессе разработки основной системы осуществлять *одновременно и согласованно* разработку оконного интерфейса (формат **GUI – G**raphical **U**ser **I**nterface), учитывая при этом парадигму объектно-ориентированного проектирования >> см. материалы к ПЗ **№8.**

12. etc.

П р и м е ч а н и е. Указанная последовательность является примерной. По мере осуществления последующих этапов возникает необходимость уточнять реализацию предыдущих, то есть реализуется *итеративная технология* разработки Windows-приложения.

Одной из предлагаемых тем курсовых работ является, *рассмотренное на практических занятиях*, **Моделирование работы банка**. *В курсовой работе* студенту предлагается приблизить модель к реальной системе, в частности, дополнительно выполнить: UML-диаграмму взаимодействия классов; *компонентно*-ориентированное программирование; реализацию программы в формате оконного приложения (GUI); моделирование других банковских операций, например, перечисление денег с одного счета на другой; идентификацию клиента кассиром банка и др. (см. на следующей странице Элементы графического интерфейса).

Некоторым студентам, не имеющим надлежащего опыта в компьютерных технологиях, предлагается наполнить *скелет* программы **Моделирование работы банка** содержанием других предметных областей, например, Моделирование работы отдела кадров, Моделирование работы библиотеки, Моделирование работы дома моды и других реальных систем.

Существенным и необходимым для усвоения дисциплины является самостоятельная работа студентов во *вне*аудиторное время с содержательной частью разработанных материалов, составление ответов на контрольные вопросы и разработка программ согласно упражнениям, которые приведены в заключительной части материалов к лекциям и практическим.

В докладе представлена основная часть учебно-методического комплекса. В 2007/08 учебном году будет разработана другая часть – компьютерные модели ряда реальных и концептуальных систем, которые будут изучаться на лекциях и практических занятиях по дисциплине Объектно-ориентированный анализ и программирование. Эти модели также будут размещены на сайте ГУ-ВШЭ.

В перспективе, по мере накопления опыта, целесообразно создать при кафедре Основы информатики и прикладного программного обеспечения группу, в составе преподавателей и студентов, задачей которой будет разработка и внедрение по заказам предприятий и организаций компьютерных моделей реальных систем, в том числе и на коммерческой основе. Это позволит кафедре осуществлять курсовое и дипломное проектирование по реальным темам, студентам накапливать практический опыт и определиться с местом работы еще на студенческой скамье.

Литература

1. Лафоре Р. Объектно-ориентированное программирование в С++. – СПб. : Питер, 2005.

2. Забудский Е.И. Учебно-методический комплекс дисциплины «Объектно-ориентированный анализ и программирование». М.: Кафедра ОИиППО ГУ-ВШЭ, 2007, Internet-ресурс – http://vorona.hse.ru/sites/infospace/podrazd/facul/facul_bi/koiippo/DocLib3/New_Web_Page_ZEI.doc .

3. Микелсен К. Язык программирования C#. Лекции и упражнения. – СПб.: ООО «ДиаСофтЮП», 2002.

Элементы графического интерфейса программы **Моделирование работы банка**;

Выполнил: студент группы 374(2) Бодряков Ю.А.