Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

Объектно-ориентированный анализ и программирование на языке C# (C_Sharp)

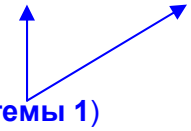
Материалы к 1-й лекции. Часть 1

Проф. Забудский Е.И.

Москва 2006

Лекция 1, часть 1

Тема 1. Основы объектно-ориентированного мышления и подхода. Класс и его составляющие в ООП, программная реализация на C#



(в **1-й части лекции** сделан акцент на первой части **темы 1**)

Уважаемые студенты!

В процессе изучения курса вы получите базовые знания по объектно-ориентированному программированию. Освоите основные принципы объектного подхода, познакомитесь с историей объектно-ориентированного программирования и с его эволюцией на базе существующих парадигм программирования. Вы усвоите основы терминологии, а также научитесь **использовать преимущества объектно-ориентированного программирования *для компьютерного моделирования реальных и концептуальных систем (в том числе бизнес-систем).***

Три основных принципа объектно-ориентированного программирования: **инкапсуляция**, **наследование** и **полиморфизм**.

На лекциях и практических занятиях, вы освоитесь с основными принципами объектно-ориентированного программирования. У вас сложится полное представление о том, что такое объектно-ориентированное программирование. Вы научитесь применять основные принципы ООП при программировании на языке **C#** (**C_Sharp**).

Вопросы и упражнения, которые будут предлагаться для выполнения дома, помогут вам лучше разобраться в пройденном материале.

Необходимым условием усвоения дисциплины

является ВАША самостоятельная работа

Уважаемые студенты! Советую Вам все материалы, подготовленные мной к лекциям и практическим занятиям, распечатать и прорабатывать их! Приведенные C#-программы реализовать в среде MS VS .NET 2005 и разобраться в них.

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены **C#** и платформа **.NET** (step by step).

Содержание

1. Вступление в объектно-ориентированное программирование

Хотя объектно-ориентированные языки употребляются с 60-х годов прошлого столетия **(см. Материалы к Практ. зан. 1, разд. 4 на с. 10 и рис.1.4)** , в последние **10** лет мы являемся свидетелями беспрецедентного роста их применения в производстве программного обеспечения. Первое время они не пользовались популярностью, однако успех таких языков, как **Java, C++** и **C#** привлек внимание к объектно-ориентированным технологиям. И это не случайно. После долгих лет забвения и тяжелой борьбы с прочно установившимися традициями, **о**бъектно-**о**риентированное **п**рограммирование (**ООП**) в своем развитии достигло того уровня, когда люди, наконец, смогли увидеть потенциальные возможности этой технологии. Сегодня во многих компаниях его употребление обязательно. Не будет преувеличением сказать, что в итоге сделаны правильные выводы.

Не стоит паниковать, если объектно-ориентированные языки (**C++, Java, C#**) вам не знакомы. Для объяснения понятий объектного подхода будем использовать язык **C#** (**C_Sharp**).

Рассмотрим следующие вопросы:

1.1. Объектно-ориентированное программирование в историческом контексте

Чтобы понять настоящее состояние ООП, надо немного знать историю программирования. ООП не было придумано за один день. Его появление является **очередной** ступенькой в естественном развитии программного обеспечения. Со временем стало проще определить, какие методологии удобны для работы, а какие нет. ООП эффективно сочетает в себе наиболее удачные, проверенные временем методологии.

Новый термин	Буквы **ОО** в **ОО**П означают "**О**бъектно-**о**риентированный подход". Этот подход включает в себя все стилевые усовершенствования, базирующиеся на понятии "**объект**" — понятии, которое многие люди обозначают словом "**вещь**". Всякая вещь, простите **объект, характеризуется своими свойствами и поведением**. Объектно-ориентированный подход можно применять: **1)** как к программированию, так и **2)** к анализу и **3)** дизайну.

Можно сказать, что объектно-ориентированный подход — это **способ мышления**, а также способ восприятия окружающего мира с точки зрения теории объектов.

Короче говоря, объектно-ориентированный подход включает в себя все то, что можно описать в терминах теории объектов.

1.2. Предшественники объектно-ориентированного программирования

Когда вы работаете с компьютером, вы пользуетесь достижениями информатики, накопленными в течение **пятидесяти** лет. Первоначально программирование было хитроумным изобретением, которое позволило программистам вводить программы через коммутационный блок напрямик в основную память компьютера. Программы писались на машинных языках в двоичном представлении. При написании программ на машинных языках часто допускались ошибки, причем из-за невозможности их структурировать сопровождение кода было практически невозможным. Кроме того, программа в машинных кодах была сложна для понимания (**парадигма программирования на машинном языке (1-я)**).

Со временем компьютеры стали применяться более широко, и появились процедурные языки более высокого уровня; первым из них был **FORTRAN**. Однако основное влияние на развитие объектно-ориентированного подхода оказали процедурные языки, которые появись позднее, такие как **ALGOL**. Процедурные языки дают программисту возможность разбить программу обработки информации **на несколько процедур более низкого уровня**. Такие процедуры более низкого уровня определяют структуру программы в целом. Последовательные обращения к этим процедурам управляют выполнением программы, состоящей из процедур. Программа прекращает работу тогда, когда исчерпывается список вызовов процедур (**парадигма процедурного программирования (2-я)**).

Эта новая парадигма программирования была прогрессивной по сравнению с парадигмой программирования на машинном языке, в нее было добавлено главное средство структурирования — процедуры. Более мелкие функции не только проще понять, но также проще отладить. Но, с другой стороны, процедурное программирование **ограничивает** *повторное* использование кода. Притом очень часто программисты писали "макаронные" программы — программы, выполнение которых напоминало распутывание макарон в тарелке спагетти. И, в конце концов, стало понятно, что концентрация внимания **на данных** при разработке программ методами процедурного программирования сама по себе **вызывает проблемы**. Так как **данные и процедура разделены, нет инкапсуляции данных.** Это приводит к тому, что каждая процедура должна знать, что делать с данными. К сожалению, когда процедура ведет себя плохо, она может испортить данные, если выполнит неправильные действия над ними (**см. Материалы к Практ. зан. 1, рис. 1.1 на с. 5**). Поскольку каждая процедура должна была дублировать информацию о доступе к данным, изменение представления данных приводило к изменениям всех тех мест программы, в которых этот доступ осуществлялся. Таким образом, даже маленькая поправка могла привести к целому ряду изменений во всей программе — другими словами **к кошмару при сопровождении** системы программного обеспечения.

В модульном программировании с такими языками, как **Modula2**, была сделана попытка устранить некоторые недостатки, найденные в процедурном программировании. Модульное программирование разбивает программу на ряд составляющих компонентов, или модулей. В отличие от процедурного программирования, которое разделяет данные и процедуры, модули объединяют их (**инкапсулируют**) (**см. Материалы к Практ. зан. 1, рис. 1.2 на с. 6**). **Модуль состоит из самих данных и процедур для обработки данных**. Когда другим частям программы нужно использовать модуль, они просто обращаются к интерфейсу модуля. Модули скрывают всю внутреннюю информацию от остальных частей программы. Основываясь на этом, довольно просто объяснить, что такое состояние: модуль сохраняет информацию о состоянии, причем эта информация может измениться (**парадигма модульного программирования (3-я)**).

Новый термин	**Состояние объекта** — это совокупность значений внутренних переменных объекта.

Новый термин	**Внутренняя переменная** — это величина, хранимая внутри объекта.

Однако **модульное программирование имеет свои недостатки**. Модули не расширяемы, это означает невозможность производить пошаговые изменения модуля без непосредственного доступа к коду и его прямого изменения. Кроме того, при разработке одного модуля нельзя использовать другой, иначе как через передачу (делегирование) функций. И хотя в модуле можно определить тип, один модуль не может использовать тип, определенный в другом модуле.

В модульных и процедурных языках у структурированных и неструктурированных данных есть свой "тип". Тип легче всего определить **как формат данных в оперативной памяти**. В языке со строгим контролем типов каждый объект должен иметь конкретный определенный тип.

2. Парадигма Объектно-Ориентированного Программирования (4-я)

ООП занимает следующую логическую ступень после модульного программирования. ООП добавляет к модулю: **1) наследование** и **2) полиморфизм**. Применяя ООП, программист структурирует программу, разделяя ее на ряд высокоуровневых объектов. Каждый объект моделирует определенный аспект решаемой проблемы. Объектно-ориентированное программирование уже не концентрирует внимание программиста на составлении последовательного списка вызовов процедур для управления процессом выполнения программы. Вместо этого объекты взаимодействуют между собой. Программа, разработанная с помощью объектно-ориентированного подхода, представляет собой действующую **модель решаемой проблемы**.

2.1. Объектно-ориентированный подход к созданию программ

Представьте, что вы разработали с помощью объектно-ориентированного подхода программу моделирования в реальном масштабе времени торгового терминала или тележки для магазинов самообслуживания. Программа, разработанная по методологии ООП, будет содержать следующие объекты:

- **item** (товар),
- **shopping cart** (тележка для магазинов самообслуживания),
- **coupon** (купон),
- **cashier** (кассир).

Взаимодействуя между собой, эти объекты управляют программой. Например, когда кассир подсчитывает стоимость заказа, он выписывает чек на сумму, соответствующую цене каждого товара.

Определение программы в терминах объектов — это наиболее понятный способ разработки программного обеспечения. Объекты заставляют вас воспринимать все с той точки зрения, **что объект делает, т.е. мысленно моделировать его поведение**. Благодаря этому вы можете отвлечься от рассмотрения объекта с точки зрения того, как он будет реализован в процессе исполнения программы. Таким образом, в процессе написания программы можно использовать естественные термины реального мира. Вместо того чтобы строить программу в форме отдельных процедур и данных (т.е. в терминах мира компьютеров), вы строите свою программу из объектов. Объекты позволяют в программе моделировать реальный мир с помощью: **1)** существительных, **2)** глаголов **3)** и прилагательных, взятых из **предметной** области.

Новый термин	Реализация определяет то, **как выполняются действия**. В терминах программирования **реализация — это программный код**.

Новый термин	Предметная область, или домен — это **абстрактное пространство**, в котором формулируется определенная задача, т.е. набор понятий, представляющих важные аспекты решаемой задачи

Рассуждая в терминах решаемой задачи, можно избежать опасности увязнуть в деталях реализации. Конечно, некоторые высокоуровневые объекты должны взаимодействовать с компьютером, пользуясь низкоуровневыми, машинно-ориентированными методами. Однако объект изолирует это взаимодействие от остальной части системы.

Примечание	В примере с тележкой для магазинов самообслуживания сокрытие реализации означает, что кассир при подсчете стоимости заказа не смотрит на исходные данные. Кассир не может обратиться к ячейкам массива, в которых хранятся числовые данные, характеризующие единицу товара, и другие переменные талона. Вместо этого кассир взаимодействует с объектом **item** (**товар**). Кассир знает, как запросить у объекта **item** (**товар**) его стоимость.

Теперь можно дать определение понятию **объект**:

Новый термин	**Объект** — это конструкция программы, в которой инкапсулировано **состояние** и **поведение**. С помощью объектов можно строить программу, используя **термины реального мира и абстракции**.

Строго говоря, **объект — это экземпляр класса**. В следующей разделе будет введено используемое в ООП понятие класса.

Объектно-ориентированная программа, как и реальный мир, состоит из объектов. В объектно-ориентированном языке программирования **C#** все является объектом, начиная от самых первичных, базовых типов, целых, логических и до наиболее сложных экземпляров классов.

2.3 Что такое класс?

Подобно реальным, объекты ООП классифицируются: **1)** по их свойствам **2)** и поведению.

В биологии собаки, коты, слоны и люди относятся к млекопитающим. Объединяют этих разных существ общие свойства. Таким же образом в мире программного обеспечения объекты относятся к одному или разным классам (**см. Материалы к Практ. зан. 1, рис. 1.3 на с. 9**).

Объектам одного класса присущи общие свойства. Иными словами, класс определяет**: 1)** свойства и **2)** поведение, *характеризующие* объект, а также, что особенно важно, **3)** те сообщения, на которые *отвечает* объект. Когда один объект оказывает влияние на поведение другого объекта, он не делает это непосредственно, а просит другой объект изменить себя, используя некоторую дополнительную информацию. Обычно это называется "посылка сообщения".

Новый термин	Класс объединяет объекты с **общими свойствами** и **поведением**. Объекты, относящиеся к одному классу, имеют одинаковые свойства и характеризуются одинаковым поведением.

Классы подобны шаблону или резальной машине для домашнего печенья**: они используются для создания экземпляров объектов**.

Новый термин	Признаки — это **видимые извне свойства класса**. Примерами признаков могут быть цвет глаз или волос.

Объект обнаруживает признаки тогда, когда предоставляет непосредственный доступ к внутренней переменной или возвращает значение с помощью метода.

Новый термин	Поведение — **это действия**, выполняемые объектом **в ответ на сообщение** или на изменение состояния. Это то, что объект делает.

Один объект может влиять на поведение другого объекта, выполняя действия над ним. Вместо термина действие употребляют термины: вызов метода, вызов функции или передача сообщения. Важно, конечно, не то, какой из этих терминов употребляется, а то, что эти действия вызывают проявление поведения объекта.

Новый термин	То, какой из этих терминов: передача сообщения, действие, вызов метода или вызов функции, используется наиболее часто, зависит от прошлого опыта разработчика.

Термин **передача сообщения** наиболее близко передает суть объектно-ориентированного подхода. Этот термин отражает динамичность взаимодействия объектов. Он подчеркивает разницу между сообщением и объектом. С его помощью легче понять суть взаимодействия объектов.

Будет в основном применять термин вызов метода. Он может быть взаимозаменяем с термином сообщение.

2.4. Соединяем все вместе: классы и объекты

Возьмем, к примеру, объект **item** (товар). У товара есть обозначение (**description**), идентификатор (**id**), штучная цена (**unit price**), количество (**quantity**) и необязательная скидка (**optional discount**). Товар должен знать, как высчитать цену со скидкой (**discounted price**).

В мире ООП все объекты **item** (товар) являются экземплярами класса **Item**. Класс **Item** может выглядеть примерно так (**см. далее строки 10…93**):

```
01  using System;       // Листинг 5. Программа на объектно-ориентированном языке C#
02  namespace ConAppl_OOP21_c32_35
03  { /*
04  *  В мире ООП все объекты item (товар) являются экземплярами класса Item.
05  *  Класс Item представляет товар так, как его видит покупатель,
06  *   когда может положить его в тележку для магазинов
07  *  самообслуживания или когда оплачивает его с помощью кассового
08  *   аппарата. На примере класса Item демонстрируется понятие класса в ООП.
09  */
10  public class Item              // начало КЛАССА Item  ////////////////////////////////////////////////////////
11  {                              // Характеристики товара:
12  private double unit_price;  // 1) цена единицы товара,
13  private double discount;    // 2) скидка с цены в процентах; скидки может не быть,
14  private int    quantity;        // 3) количество товара,
15  private String description; // 4) описание товара,
16  private String id;              // 5) код товара.
17
18  public Item(String id, String description, int quantity, double price)
19  { // начало КОНСТРУКТОРА Item() класса Item
20   this.id = id;
21   this.description = description;
22   if( quantity >= 0 )
23  {
24     this.quantity = quantity;
25  }
26  else
```

```
27  {
28      this.quantity = 0;
29  }
30      this.unit_price = price;
31  } // окончание КОНСТРУКТОРА Item() класса Item
32
33  public double Adjusted_Total()
34  {  // в этом МЕТОДе выполняется расчет итоговой цены приобретаемого товара
35      double total = unit_price * quantity;
36      double total_discount = total * discount;
37      double adjusted_total = total - total_discount;
38      return   adjusted_total;
39   }
40
41  public double Discount   //СВОЙСТВО Discount : set/get процентную скидку с цены
42  {
43  set
44  {
45      if( discount <= 1.00 )
46      {
46          this.discount = value;                  // discount = 0.15
47      }
48      else
49      {
50      this.discount = 0.0;
61      }
62  }
63  get
64  {
65      return discount;
66  }
67  }
68
69      public int Quantity    // СВОЙСТВО Quantity : set/get количество товара
70      {
71      set
72      {
73          if (quantity >= 0)
74          {
75              this.quantity = value;
76          }
```

```
77      }
78      get
79      {
80          return quantity;
81      }
82  }
83
84      public String Id    // СВОЙСТВО Id : get код товара
85      {
86          get { return id; }
87      }
88
89      public String Description  // СВОЙСТВО Description : get описание товара
90      {
91      get {return description;}
92      }
93  } ////////////////////////////// конец КЛАССА Item ////////////////////////////////////////////////////////////////////////////
94
95  public class ItemExample  ///////////////// Начальный класс ItemExample ////////////////////////////
96  { // Функция Main демонстрирует создание объектов и работу с ними.
97  public static void Main( String [ ] args )  ////////////////////////////////////////////////////////////////////////////////
98  {
99      // конструктор Item() создает товары (ОБЪЕКТЫ) класса Item, которые
100     // имеют след-е атрибуты: код, описание, количество, цена за единицу
101     Item milk     = new Item("молочный-O1I",  "Молоко (литр)    ",      2,  2.50);
102     Item yogurt = new Item("молочный-032",  "Персиковый йогурт", 4,  0.68);
103     Item bread  = new Item("пекарня-023",      "Нарезанный хлеб  ",   1,  2.55);
104     Item soap    = new Item("для дома-21",      "6 упаковок мыла  ",    1,  4.51);
105
106 // применение дисконтной карты - предоставлена скидка только на стоимость
    молока
107 milk.Discount = 0.15 ;
108
109 // получаем цены на товары, откоректир-ные с учетом скидки и количества товара
110 double     milk_price     =     milk.Adjusted_Total();
111 double yogurt_price     = yogurt.Adjusted_Total();
112 double  bread_price     =   bread.Adjusted_Total();
113 double   soap_price     =    soap.Adjusted_Total();
114
115 // напечатать квитанцию
116 Console.WriteLine("Благодарю Вас за ваши покупки.");
117 Console.WriteLine("Пожалуйста, приходите снова!\n");
118 Console.WriteLine(milk.Discount + "\t\t" + milk.Quantity +
```

```
119                                                  "\t\t" + milk.Description + "\t $" + milk_price);
120  Console.WriteLine(yogurt.Id + "\t\t\t" + yogurt.Description + "\t $" + yogurt_price);
121  Console.WriteLine(bread.Discount + "\t\t\t\t" + bread.Description + "\t $" + bread_price);
122  Console.WriteLine("\t\t\t\t" + soap.Description   + "\t $" + soap_price);
123
124  // подсчитать и напечатать сумму - стоимость всей покупки
125  double total = milk_price + yogurt_price + bread_price + soap_price;
126  Console.WriteLine("\n\t\t\t\tОбщая цена товаров     \t $" + total);
127  Console.ReadLine();
128  } /////////////////////////////////////// конец метода Main ////////////////////////////////////////////////////////////
129  } ////////////////////////////////////// конец класса ItemExample ////////////////////////////////////////////////////////
130  }
```

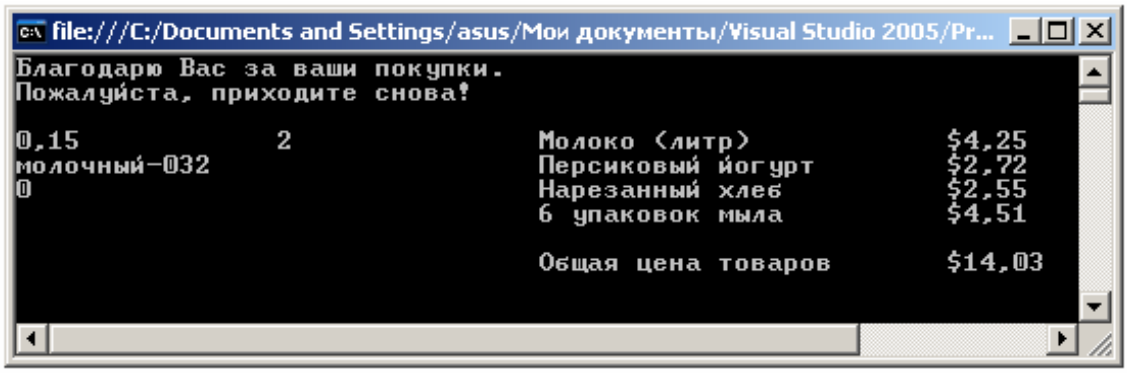


Рис. 1.1. Вывод квитанции

Такие методы, как (**см. выше строки 18…31**)

public Item(String id, String description, int quantity, double price)

называются конструкторами. Конструкторы инициализируют объект в процессе его создания (**см. выше строки 99…104**).

Два новых термина

Конструкторы — это методы, используемые для инициализации объектов в процессе их реализации. Создание объекта называется реализацией, потому что в его процессе **создается объект — экземпляр класса** (**см. Материалы к Практ. зан. 2, Доп. 1, с. 4…11**).

В конструкторе и в экземпляре **Item** используется **this**;

this — это ссылка, указывающая на объект-экземпляр. У каждого объекта есть ссылка на себя самого. Экземпляр использует эту ссылку для того, чтобы получить доступ к своим переменным и методам.

Свойства **Discount()** (**см. выше с. 41…67**), **Description()** (**см. выше с. 89…92**) и метод **Adjusted_Total()** (**см. выше с. 33…39**) реализуют поведение класса **Item**. Вызов этих методов возвращает или устанавливает параметры. Когда кассир подсчитывает стоимость заказа, т.е. общую стоимость содержимого тележки в магазине самообслуживания, он берет каждый товар и посылает объекту сообщение **Adjusted_Total()** (**см. выше с. 109…113**).

unit_price, **discount**, **quantity**, **description** и **id** (**см. выше с. 12…16**) являются внутренними переменными класса **Item**. В их значениях содержится информация о состоянии объекта. Состояние

объекта может изменяться со временем. Например, во время посещения магазина с целью покупки покупатель может приложить к товару купон, что изменит состояние товара, так как изменится значение **discount**.

Способы поведения, реализуемые методом **Adjusted_Total()** и свойством **Discount()** называются средствами доступа, так как они предоставляют доступ к **внутренним** (**private**) данным объекта (**см. выше с. 12…16**). Доступ может быть прямым, такой доступ реализован в свойстве **Discount()**. Но иногда объект должен предварительно произвести некоторые вычисления, чтобы выдать необходимую информацию, как в методе **Adjusted_Total()**.

новый термин	С помощью средств доступа можно получить доступ к **внутренней** (**private**) информации объекта, однако нельзя узнать, находится ли информация в переменной, в комбинации переменных или вычисляется.

Средства доступа позволяют изменять (**set**) или считывать (**get**) значение (**см. Материалы к Практ. зан. 2, Доп. 1, с. 12…18**), а также могут приводить к побочным эффектам, влияя на внутреннее состояние.

Во время выполнения программа использует классы, такие как **Item**, для создания или реализации объектов, выполняющих задачу. Каждый новый экземпляр — это дубликат класса **Item**. Однако реализованный экземпляр характеризуется своим поведением (т.е. выполняет свои собственные функции) и отслеживает свое собственное состояние. Таким образом, то, что создается как клон (дубликат, абсолютная копия), может вести себя по-своему во время своего существования.

Например, если вы создадите два объекта-товара как экземпляры одного и того же класса **Item**, у одного элемента может быть скидка к цене **10%,** тогда как у другого скидки к цене может не быть вообще. Некоторые товары стоят дороже, чем другие. Один товар может стоить **1 000** долларов, а дугой — **1,98** доллара. И хотя состояние товара может меняться со временем, он все равно будет оставаться экземпляром класса **Item**. Вспомните биологический пример: серые млекопитающие являются млекопитающими в такой же степени, как и коричневые.

2.5. Запускаем объекты в работу

Рассмотрим следующий метод **Main()** (**см. выше с. 97…126**).

Этот метод показывает, как выглядит небольшая программа, использующая класс **Item**. Сначала программа реализует четыре товара класса **Item** (**см. выше с. 101…104**). Настоящая программа могла бы создавать эти товары во время просмотра пользователем каталога в диалоговом режиме или во время сканирования кассиром штрих-кодов товаров.

Эта программа создает ряд товаров, применяет скидки к цене (**см. выше с. 106…113**), а затем печатает квитанцию (**см. выше с. 115…122**). Программа демонстрирует все взаимодействия объектов, посылая сообщения товарам. Например, программа применяет **15%** скидку к цене молока, посылая сообщение **Discount()** данному товару (**см. выше с. 107**).

И, наконец, программа выводит квитанцию на экран. На **рис. 1.1** показан пример вывода данных.

Важно помнить о том, что вся программа была написана полностью в терминах, определенных в классе **Item** с помощью поведения, реализуемого методами класса **Item**. Короче говоря, программа написана с помощью **существительных** и **глаголов**, являющихся терминами в **предметной** области "тележка для магазинов самообслуживания".

2.6. Связь между объектами

Связь объектов — это важная составляющая ООП. Существует **два** основных способа связи объектов.
Первый способ: объекты существуют **независимо** один от другого. В тележке для магазинов самообслуживания в одно и то же время могут находиться два товара — экземпляра класса **Item**. Если этим отдельным объектам нужно взаимодействовать, они посылают друг другу сообщения.

Новый термин	Объекты общаются друг с другом при помощи сообщений. Получив сообщение, объект выполняет определенные действия.

Передать **сообщение** — это то же самое, что вызвать метод с целью изменения состояния объекта или применить одну из моделей поведения.

Второй способ: **один** объект может содержать **другие** объекты. Так же как в ООП программа состоит из объектов, так и объекты могут складываться из других объектов **с помощью *агрегации***. Из примера класса **Item** можно заметить, что один объект содержит много других объектов. Например, этот объект (класс **Item**) содержит объекты **description** и **id,** которые относятся к типу **String**. У каждого из этих объектов есть интерфейс, содержащий методы и признаки. Следует помнить, что **в ООП все является объектом**, **даже составные части объекта!**

Передача информации между объектами и теми объектами, которые они содержат, происходит таким же образом, как и в других случаях. Когда объекты взаимодействуют, **они посылают друг другу сообщения**.

Сообщение — это важное понятие объектно-ориентированного подхода. Благодаря механизму сообщений **объекты могут сохранять свою независимость**. Объекту, который посылает сообщение другому объекту, безразлично, как этот другой объект выполнит требуемое действие. Объекту важно лишь, чтобы действие было выполнено.

Новый термин	Вопрос о том, как наиболее правильно определить понятие объекта, все еще обсуждается. Существует мнение, что объект не следует определять как экземпляр класса. Вместо этого все определяется как объект**:** с этой точки зрения **класс — это объект, который создает другие объекты**. Мы будем употреблять термин: “**объект – экземпляр класса**”.

Сталкиваясь с расхождениями в терминологии, мы выбираем одно из определений, а затем придерживаемся его. Выбор часто обусловлен практическими соображениями. **Мы выбрали определение объекта как экземпляра класса**. Это определение заимствовано из **U**nified **M**odeling **L**anguage (**UML** – унифицированный язык моделирования), оно наиболее часто встречается в промышленности. (Несколько позднее вы более подробно узнаете об **U**nified **M**odeling **L**anguage – **см. Материалы к Практ. зан. 3, с. 26…29**).

2.7. Как в ООП используются достижения более ранних технологий

Так же как и при создании других парадигм программирования, парадигму ООП разрабатывали с учетом всех сильных сторон предшествующих парадигм, стараясь избежать их недостатков. Поэтому в ООП учтены все достижения: **1) процедурного 2)** и **модульного** программирования.

В модульном программировании программа состоит из ряда модулей. Подобно этому в ООП программа разбивается на ряд взаимодействующих объектов. Так же как **модули *упрятывают* представление данных в процедурах**, объекты **инкапсулируют** свое состояние, **скрывая его**

за интерфейсом. Понятие **инкапсуляции** ООП напрямую заимствовало из модульного программирования.

Инкапсуляция очень сильно отличается от процедурного программирования. *В процедурном программировании данные **не** инкапсулируются*. Наоборот, все процедуры имеют доступ к данным. В отличие от процедурного программирования в объектно-ориентированном программировании **данные и поведение прочно соединены в объекте**. Подробнее об инкапсуляции будет рассказано на следующей лекции.

Хотя в своей основе объекты похожи на модули, у них есть много важных отличий. Во-первых, модули трудно расширять. Чтобы устранить этот недостаток, в объектно-ориентированном программировании введено понятие **наследования**. Наследование позволяет легко расширять и увеличивать классы. Наследование также позволяет классифицировать классы. О наследовании вы подробнее узнаете на третьей лекции.

В ООП также усилена концепция **полиморфизма**; благодаря этому облегчается создание гибких, легко изменяемых программ. Именно концепция полиморфизма позволяет придать программам гибкость, устраняя присущие модульному программированию ограничения, связанные с жестким контролем типов. О полиморфизме вы подробнее узнаете на четвертой лекции. Безусловно, ни инкапсуляция, ни полиморфизм не были изобретены в ООП. Однако именно ООП объединило в себе эти две концепции. Добавив концепцию объекта из ООП, вы получите неизвестную ранее комбинацию этих технологий программирования.

3. Преимущества и цели объектно-ориентированного подхода

ООП преследует шесть основных целей в разработке программного обеспечения. Программное обеспечение, разработанное в соответствии с **парадигмой ООП**, должно иметь следующие свойства:

1) естественность;
2) надежность;
3) возможность повторного использования;
4) удобство в сопровождении;
5) способность совершенствоваться;
6) удобство периодического выпуска (издания) новых версий.

Рассмотрим, какие преимущества дает каждая из этих характеристик.

3.1. Естественность

С помощью ООП создается естественное программное обеспечение. Естественные программы более понятны. Вместо того чтобы использовать при программировании такие термины, как массив или область памяти, можно использовать **терминологию из той предметной области, к которой принадлежит решаемая задача**. При создании программы не нужно вникать во все детали, связанные с компьютером. Вместо того чтобы подгонять разрабатываемую программу под компьютерный язык, ООП дает возможность пользоваться терминологией конкретной предметной области.

Используя объектно-ориентированное программирование, можно смоделировать решение задачи на **функциональном** уровне, а **не** на уровне реализации. Для того чтобы использовать определенную составляющую программы, нет необходимости знать, как она работает: вы **сосредотачиваетесь на том, что она делает.**

3.2. Надежность

Хорошее программное обеспечение должно быть таким же надежным, как и другие изделия, например, холодильники или телевизоры.

Хорошо разработанная и аккуратно написанная объектно-ориентированная программа очень надежна. Модульная природа объектов позволяет производить изменения **в одной из частей** программы, **не затрагивая других ее частей**. Благодаря понятию объекта, информацией владеют именно те, кому она нужна, а ответственность возлагается на тех, кто выполняет данные функции.

Тщательная проверка увеличивает надежность. Используя объектно-ориентированный подход, информацию и ответственность можно сосредоточить в одном месте, а значит, и изолировать от других частей системы. А это, в свою очередь, позволяет расширить возможности проверки. Именно благодаря изолированности можно проверить каждый компонент в отдельности. Проверив компонент, вы можете спокойно использовать его повторно.

3.3. Повторное использование

Строитель не изобретает новый вид кирпичей каждый раз, когда собирается строить дом. Инженер-электротехник не придумывает новый вид резисторов каждый раз, когда создает новую схему. Так почему же программист продолжает "изобретать колесо"? Если задача решена, нужно многократно использовать ее решение.

Профессионально разработанные объектно-ориентированные классы можно смело использовать повторно. Так же как и модули, объекты можно повторно использовать в различных программах. В отличие от модульного программирования, ООП для расширения существующих объектов позволяет использовать **наследование**, а для написания настраиваемого кода — **полиморфизм**.

Применение объектно-ориентированного подхода не дает гарантии создания настраиваемого кода. Создание хороших классов — это занятие для виртуозов-профессионалов, требующее искусства, доступного подчас лишь профессионалам самого высокого класса, и внимательного подхода к разного рода абстракциям при выделении основных признаков. Программисты хорошо знают, что овладеть этими качествами не просто.

С помощью ООП можно воплощать довольно абстрактные идеи и использовать их для решения конкретных задач.

3.4. Сопровождение

Жизненный цикл программного изделия не заканчивается после его разработки. В процессе эксплуатации программы необходима поддержка, которая называется сопровождением. От 60% до 80% времени, потраченного на программу, уходит на сопровождение. Разработка составляет всего лишь 20% жизненного цикла.

Хорошо разработанная объектно-ориентированная программа **удобна в обслуживании**. Чтобы устранить ошибку, нужно внести исправление **только в одно** место. Так как изменение реализации прозрачно, все другие объекты автоматически начинают пользоваться преимуществами усовершенствования. Благодаря своей естественности текст программы должен быть понятен для других разработчиков.

3.5. Способность к расширению

Во время сопровождения программы пользователи часто просят добавить к системе новые функции. Да и по мере создания библиотеки объектов приходится расширять функциональность этих

объектов.

ООП учитывает эти реалии. Программное обеспечение не статично. Для того чтобы программное обеспечение оставалось полезным, нужно постоянно увеличивать его возможности. В ООП есть много способов для расширения программы – среди них **1)** наследование, **2)** полиморфизм, **3)** переопределение, **4)** делегирование **5)** и множество шаблонов, которые можно использовать в процессе разработки.

3.6. Периодический выпуск (издание) новых версий

Жизненный цикл современного программного изделия часто измеряется неделями. Благодаря ООП цикл разработки программ удалось сократить, так как программы стали более надежными, легче расширяются и могут использоваться повторно.

Естественность программного обеспечения упрощает разработку сложных систем. Любая разработка требует тщательного подхода, поэтому естественность позволяет сократить циклы разработки программного обеспечения, поскольку дает возможность сосредоточиться на решаемой задаче.

После того, как программа разбита на ряд объектов, разработку каждой отдельной части программы можно вести параллельно с другими. Несколько разработчиков могут разрабатывать классы независимо друг от друга. Такой параллелизм в разработке сокращает ее время.

4. Ошибки

Начинающие изучать объектно-ориентированный подход часто допускают одни и те же четыре ошибки.

4.1. Ошибка 1: ООП — простой язык

Некоторые люди полагают, что ООП и объектно-ориентированные языки — это одно и тоже. Полагая так, вы будете уверены в том, что применяете в программировании объектно-ориентированный подход, уже хотя бы потому, что используете объектно-ориентированный язык C# (C_Sharp). Нет ничего более далекого от правды.

ООП предполагает намного больше, чем простое использование объектно-ориентированного языка или знание набора определений. Можно написать абсолютно не объектно-ориентированную программу на объектно-ориентированном языке. **Настоящее ООП — это образ мышления, заставляющий рассматривать проблему в виде группы объектов и правильно использовать инкапсуляцию, наследование и полиморфизм**.

К сожалению, многие компании и программисты полагают, что если они используют объектно-ориентированный язык, у них будут все преимущества ООП. Когда их надежды не оправдываются, они обвиняют используемую технологию, не учитывая того факта, что это именно они не обучили должным образом своих подчиненных или же польстились на популярный метод программирования, не понимая до конца его принципов.

4.2. Ошибка 2: страх повторного использования (см. Материалы к Практ. зан. 4, с. 21…36)

Нужно научиться повторно использовать программу. Когда вы начинаете работать с ООП, обучение безошибочному повторному использованию является одним из наиболее трудных заданий.

Во-первых, программистам нравится творить и поэтому создается впечатление, что повторное использование лишает вас удовольствия, присущего самостоятельному творческому процессу созидания программы. Однако следует помнить, что при повторном использовании ранее разработанных элементов вашей целью является создание чего-то большего, чем те элементы, которые вы используете повторно. Повторное использование элемента может показаться не очень

заманчивым, однако именно оно позволяет создать нечто лучшее.

Во-вторых, многие программисты не доверяют тому, что было написано не ими. Если часть программного обеспечения хорошо проверена и подходит вам, вы не должны бояться использовать ее повторно. Не отказывайтесь от какого-либо компонента только потому, что его писали не вы. Помните, что повторное использование позволит вам написать еще одно замечательное программное обеспечение.

4.3. Ошибка 3: ООП — лекарство от всех болезней

Хотя ООП обладает множеством преимуществ, оно вовсе не является лекарством от всех болезней в мире программирования. В некоторых случаях вы **не должны использовать объектно-ориентированный подход**. Нужно уметь выбрать правильное средство для выполнения определенной работы. Важно понимать, что применение ООП **совсем не гарантирует** успех какого-либо проекта. Успех приходит только тогда, когда **планирование**, **разработка** и **программирование** выполнены самым тщательным образом.

4.4. Ошибка 4: эгоцентрическое программирование

Программируя, не следует быть эгоцентристом. Уметь делиться создаваемой программой так же необходимо, как и уметь повторно использовать чужие. Делиться означает создавать удобные для использования другими разработчиками классы, а также упрощать повторное использование этих классов.

Когда программируете, **помните о других** разработчиках. Создайте четкий, понятный интерфейс. Что наиболее важно, пишите документацию. Документируйте допущения (предусловия), документируйте параметры метода, документируйте как можно больше. Люди не будут повторно использовать то, что не возможно найти или понять.

Резюме

Вы ознакомились с объектно-ориентированным программированием. Было **рассмотрено развитие основных парадигм программирования**, а также некоторые базовые понятия ООП. К этому моменту вы должны иметь представление о таких важных понятиях объектно-ориентированного подхода, **как класс и о том, как объекты обмениваются информацией**.

Определения важны, однако не следует, застряв в методах решения, забывать о той задаче, которую мы пытаемся решить с помощью объектно-ориентированного подхода (**задача компьютерного моделирования реальных и концептуальных систем, в том числе бизнес-систем**). С помощью объектно-ориентированного программирования мы хотим достичь шести основных целей (их же можно будет рассматривать в качестве преимуществ программ, разработанных с помощью объектно-ориентированного подхода):

1. естественность;
2. надежность;
3. возможность повторного использования;
4. удобство в сопровождении;
5. способность к расширению;
6. удобство периодического выпуска (издания) новых версий.

Никогда не забывайте об этих целях.

Контрольные вопросы

1. Что такое процедурное программирование?
2. Какими преимуществами обладает процедурное программирование по сравнению с неструктурным программированием?
3. Что такое модульное программирование?
4. Какими преимуществами обладает модульное программирование по сравнению с процедурным программированием?
5. Перечислите недостатки процедурного и модульного программирования.
6. Что такое объектно-ориентированное программирование?
7. Каковы шесть преимуществ и целей объектно-ориентированного программирования?
8. Объясните одну из целей объектно-ориентированного программирования.
9. Дайте определения следующих терминов:
   - класс;
   - объект;
   - поведение.
10. Как объекты обмениваются информацией?
11. Что такое **конструктор**?
12. Что такое **средство доступа**?
13. Что такое **this**?

СОВЕТ	Продумайте ответы самостоятельно – проверьте себя. Правильные ответы приведены на следующей странице.

Ответы на контрольные вопросы

1. Процедурное программирование как методика разработки программного обеспечения **разделяет** программу на данные и процедуры, **обрабатывающие эти данные.** Процедурное программирование имеет последовательную природу. В процессе выполнения процедурной программы **последовательно вызываются различные процедуры**. После выполнения последней из них программа завершает свою работу.

2. Процедурное программирование задает общую структуру программы: **1)** данные **2)** и процедуры. Использование процедур помогает разрабатывать программу, предназначенную для решения конкретной задачи. Вместо написания одного большого блока обработки данных в ходе разработки программа все время разбивается на процедуры и более мелкие подпрограммы. Кроме того, **процедуры можно использовать повторно**. Можно даже создавать библиотеки многократно используемых процедур.

3. Модульное программирование **жестко связывает данные и процедуры**, обрабатывающие эти данные, в компоненты программы, называемые модулями. Модули скрывают представление данных и внутренние методы их обработки. Однако большинство модульных языков программирования дают возможность использовать модули в процедурной среде.

4. Модульное программирование скрывает детали реализации алгоритмов, **защищая, таким образом, данные от несовместимого или некорректного изменения**. Использование модулей также задает высокоуровневую структуру программы. Такой подход дает возможность разрабатывать структуру программы на концептуальном, поведенческом уровне, а не в терминах данных и процедур.

Как процедурное, так и модульное программирование **слабо поддерживают многократное использование**. Хотя процедуры можно использовать повторно, они очень зависят от обрабатываемых ими данных. **Глобальный** характер данных в процедурном программировании усложняет повторное использование процедур. Часто процедуры взаимосвязаны с другими частями программы далеко не очевидным образом.

Модули сами по себе легко использовать повторно. Можно просто взять модуль из одной программы и перенести его в другую. Тем не менее, и в этом случае есть некоторые недостатки. Так, программа может использовать непосредственно сам модуль, но не может использовать его в качестве основы для создания нового модуля.

6. Объектно-ориентированное программирование — метод разработки программного обеспечения, позволяющий при моделировании программы использовать названия объектов реального мира. Оно разбивает программу на набор взаимодействующих объектов. Объектно-ориентированное программирование является развитием модульного программирования, так как **поддерживает инкапсуляцию**. По сравнению с модульным программированием, в нем введена поддержка таких свойств объектов, как **наследование** (для расширения возможности многократного использования) и **полиморфизм** (для повышения гибкости в работе с типами данных).

7. Шесть достоинств, которыми обладают написанные с помощью объектно-ориентированного программирования программы:

- Естественность
- Надежность
- Возможность многократного использования
- Удобство в сопровождении (легкость исправления ошибок)

- Расширяемость
- Быстрота создания (своевременный периодический выпуск (издание) новых версий)

8. Объектно-ориентированное программирование естественно. Вместо того чтобы описывать задачу в терминах данных и процедур, при использовании объектно-ориентированного программирования **можно пользоваться словарем исходной задачи**. Такой подход, давая возможность использовать хорошо понятные термины, позволяет сосредоточить внимание на решении задачи, а не на деталях реализации.

9. Класс определяет атрибуты и поведение, общие для группы объектов. Можно использовать определение класса для создания экземпляров таких объектов.

Объект — **это экземпляр класса**, программа работает именно с объектами.

Объект проявляет себя **в поведении**. Можно сказать, что **поведение** объекта — э**то связь объекта с внешним миром**. Другие объекты могут использовать любое поведение объекта.

10. Объекты общаются друг с другом посредством посылки сообщений. **Посылка сообщения** — синоним терминов "выполнение метода" и "вызов процедуры".

11. **Конструктор** — **метод**, определяющий способ создания экземпляра объекта. Вызов конструктора создает объект и делает его доступным для программы.

12. **Средство доступа** — **свойство**, обеспечивающее доступ к **внутренним** (**private**) данным объекта.

13. **this** — **ссылка**, с помощью которой объект обращается **к самому себе**. Ссылка **this** дает экземпляру объекта доступ к его **переменным** и **поведениям**.

Литература к курсу

Базовый учебник

1. Мейер Б. Объектно-ориентированное конструирование программных систем. М.: Русская Редакция, 2005.

Основная

2. Буч Г., Якобсон А., Рамбо Дж. **UML**. С.-Петербург: Питер, 2006.
3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. С.-Петербург: Питер, 2006.
4. Забудский Е.И. Учебно-методические материалы по дисциплине «Объектно-ориентированный анализ и программирование». М.: Кафедра ОИиППО ГУ-ВШЭ, 2005,
   Internet-ресурс – http://new.hse.ru/C7/C17/zabudskiy-e-i/default.aspx .
5. Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML. М.: Вильямс, 2003.
6. Лафоре Р. Объектно-ориентированное программирование в C++. С.-Петербург: Питер, 2005.
7. Троелсен Э. C# и платформа .NET. С.-Петербург: Питер, 2006.
8. Синтес А. Освой самостоятельно объектно-ориентированное программирование за 21 день. Москва; С.-Петербург; Киев: Вильямс, 2002.

Дополнительная – **Internet-ресурсы**

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

Дополнительная – книги

14. Мэтт Вайсфельд. Объектно-ориентированный подход: Java, .NET, C++. М.: КУДИЦ-ОБРАЗ, 2005.
15. Дж. Кьоу, М. Джеанини. Объектно-ориентированное программирование. С.-Петербург: Питер, 2005.

З а д а н и е на д о м

(упражнение по программированию)

1. Реализовать в среде MS VS .NET 2005 и *проанализировать* программу (листинг 5), рассмотренную в **разделе 2.4 (с. 8…11).**

2. Включить в эту программу **еще один товар**. Задать **самостоятельно** для этого товара**:** обозначение (**description**), идентификатор (**id**), штучную цена (**unit price**), количество (**quantity**) и скидку (**discount**). Рассчитать цену со скидкой товара и внести соответствующую информацию в квитанцию. Вывести квитанцию на все товары на печать.