ИНФОРМАТИКА
ИНФОРМАЦИОННЫЕ СИСТЕМЫ

УДК 004.432.4

Взаимодействие объектов в объектно-ориентированном программировании

Л. А. Залогова
Пермский государственный национальный исследовательский университет
614990, Россия, Пермь, ул. Букирева, 15
zalogova.la@gmail.com

В статье излагается один из возможных способов организации взаимодействия объектов объектно-ориентированной программы, приведены фрагменты программного кода для конкретного примера. Для наглядного представления передачи сообщений между объектами описаны диаграммы их взаимодействия и последовательностей. Предложенная методика может быть использована при обучении технологии объектно-ориентированного программирования.

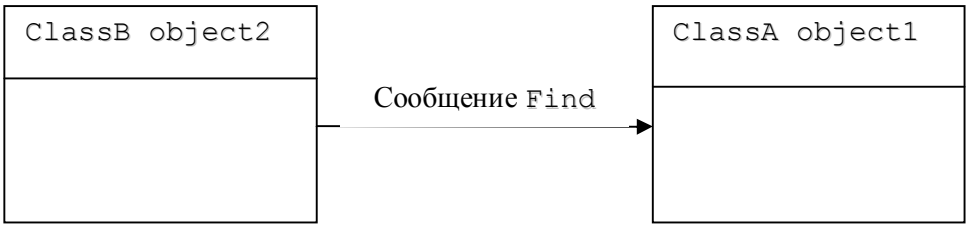
Ключевые слова: *объектно-ориентированное программирование; класс; объект; взаимодействие объектов.*

DOI: 10.17072/1993-0550-2016-3-94-102

Парадигма объектно-ориентированного программирования предполагает, что любая программная система проектируется как совокупность объектов. Обычно в литературе по объектно-ориентированным языкам описываются основные конструкции языков и примеры их использования. При этом отсутствуют содержательные задачи, в которых на основе постановки задачи выполняется объектная декомпозиция, а также демонстрируется реализация взаимодействия объектов на конкретном языке программирования, что, в свою очередь, является основной причиной трудностей в преподавании объектно-ориентированного программирования. Поэтому в статье на примере практической задачи показано, как описать предметную область в виде совокупности объектов, реализовать взаимодействие объектов и наглядно представить передачу сообщений между ними. Предложенная методика может быть использована при обучении технологии объектно-ориентированного программирования.

Как правило, использование изолированных объектов не позволяет достичь поставленной цели. Интерес представляет взаимодействие объектов, т.е. обмен информацией между объектами. Обычно говорят, что один объект взаимодействует с другим посредством сообщений, т.е. в процессе взаимодействия объекты обмениваются сообщениями. Сообщение – это вызов метода. Совместное использование методов различных объектов – показатель взаимодействия объектов. Таким образом, методы одного объекта должны вызывать методы других объектов. В этом случае ни один метод, участвующий во взаимодействии, не может достигнуть цели самостоятельно. Именно поэтому так важно правильно организовать обмен информацией между методами различных объектов. Такой обмен предполагает, что результаты работы метода одного объекта являются исходными данными для метода другого объекта. Если

объект Object2 класса ClassB посылает сообщение объекту Object1 класса ClassA, то это означает, что объект Object2 вызывает метод объекта Object1 (рис. 1). В этом случае при описании классов ClassA и ClassB возникает вопрос: как обратиться к методу класса ClassA в методе класса ClassB?



Объект Object2 вызывает метод Find объекта object1 (object1.Find)

Рис. 1. *Передача сообщения между объектами*

Решение этого вопроса состоит в следующем.

Вариант 1. Создать объект класса ClassA в качестве *поля* класса ClassB; после этого из любого метода класса ClassB можно обращаться к методам класса ClassA.

Вариант 2. Создать объект класса ClassA в качестве *локального объекта* некоторого метода класса ClassB; в дальнейшем в этом методе можно обращаться к методам класса ClassA.

Рассмотрим, как реализуется взаимодействие объектов на примере задачи "Курсы по выбору".

Постановка задачи. Каждый студент должен прослушать и сдать несколько курсов, перечисленных в учебном плане. Студент должен записаться на выбранные курсы. Только после этого можно приступить к занятиям и сдаче экзаменов. Язык программирования – C# [3].

Определим класс Student, который содержит информацию об отдельном студенте (рис. 2).

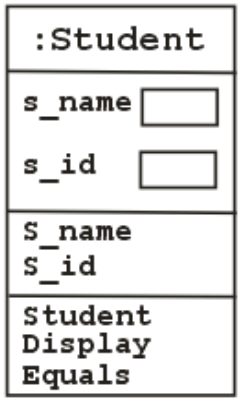


Рис. 2. *Диаграмма объекта класса Student*

В классе Student описано два поля – имя студента s_name и номер студента s_id. Методы класса: конструктор, печать информации о студенте и сравнение двух объектов класса Student на равенство (листинг 1).

Кроме класса Student необходим класс Course, который содержит информацию об отдельном курсе (рис. 3).

```
class Student{
   string s_name; // имя студента
   int s_id; // номер студента
   // описание свойств S_ name и S_id .  .  .
   public Student(string s_name, int s_id) {// конструктор .  .  . }
   public void Display(){ //вывод информации о студенте  .  .  . }
   // метод  сравнивает два объекта класса Student на равенство
   public override bool Equals(object obj) {
     Student p = obj as Student;
     return ((s_name == p.s_name)&&(s_id == p.s_id));
   }
}
```

Листинг 1. *Описание класса Student*

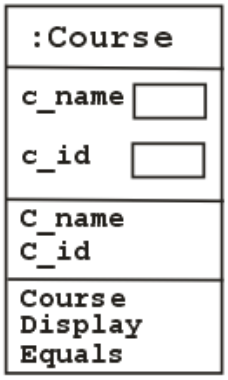


Рис. 3. *Диаграмма объекта класса* `Course`

Поля класса `Course` – название курса `c_name` и номер курса `c_id`. Методы класса – конструктор, печать информации о курсе и сравнение двух объектов класса `Course` на равенство (листинг 2).

Студент должен записаться на курсы. Однако класс `Course` не содержит информации о студентах, а класс `Student` – о курсах. Следовательно, студент не может записаться на курсы, а курс невозможно связать с информацией о студентах, которые его посещают. Поэтому опишем еще два класса `CSLink` и `CSRecords`.

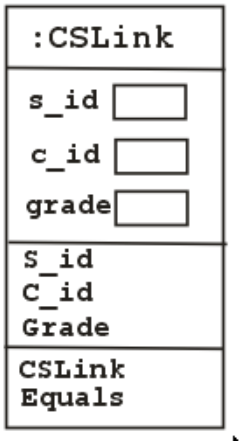


Рис. 4. *Диаграмма объекта класса* `CSLink`

Класс `CSLink` связывает одного студента с одним курсом (рис. 4).

Поля класса `CSLink` – номер студента `s_id`, номер курса `c_id` и оценка студента по курсу `grade`. Методы этого класса – конструктор и проверка на совпадение номеров студентов и номеров курсов у двух объектов класса `CSLink` (листинг 3).

Класс `CSRecords` используется для записи студентов на курсы (рис. 5).

```
class Course {
   string c_name; // название курса
   int c_id; // номер курса
   // описание свойств C_ name и C_id ...
   public Course(string c_name, int c_id){ // конструктор ...}
   public void Display(){// вывод информации о курсе ... }
   // метод  сравнивает два объекта класса Course на равенство
   public override bool Equals(object obj) { . . . }
}
```

Листинг 2. *Описание класса* `Course`

```
class CSLink {
   int s_id, c_id; // номер студента, номер курса
   int grade; // оценка
   // описание свойств S_id, C_id и Grade .  .  .
   public CSLink (int s_id, int c_id) { // конструктор  .  .  . }
   // проверка на совпадение номеров студентов и номеров курсов
   public override bool Equals(object obj) {
       CSLink p = obj as CSLink;
       return ((s_id == p.s_id) && (c_id == p.c_id));
   }
} //class CSLink
```

Листинг 3. *Описание класса* ***CSLink***

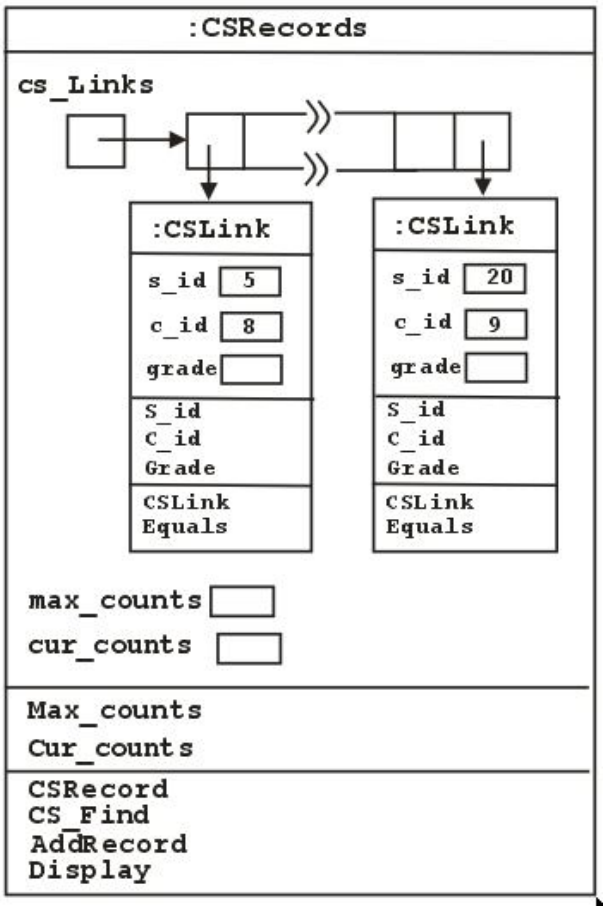


Рис. 5. *Диаграмма (пример) объекта класса* `CSRecords`

Поля класса `CSRecords` - массив `cs_Links`, содержащий информацию о том, какие студенты на какие курсы записаны; `max_counts` и `cur_counts` - максимальное и текущее количество записей студентов на курсы.

Перед записью студента на курс нужно проверить, существует ли объект с заданным номером студента и номером курса в массиве `cs_Links`. С этой целью в классе `CSRecords` описан метод `CS_Find`, который

✓ создает объект `temp` класса `CSLink` (рис. 6), содержащий заданный номер студента `num_st` и номер курса `num_course`:

```
CSLink temp =
  new CSLink(num_st, num_course);
```

✓ сравнивает объект `temp` с объектами массива `cs_Links`, а именно, вызывает метод `Equals` объекта `temp`; параметр этого метода – текущий объект массива `cs_Links`:

```
int i=0;
while((i<cur_counts)&&
      !temp.Equals(cs_Links[i]))
    i++;
```

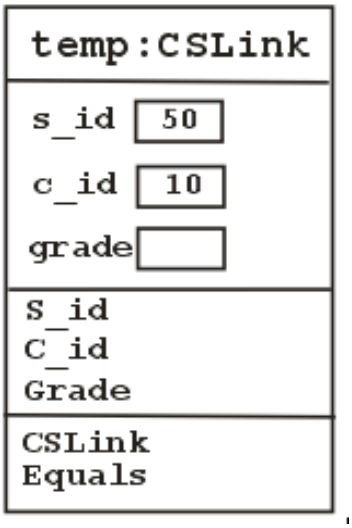


Рис. 6. *Диаграмма (пример) объекта* `temp` *класса* `CSLink`

Таким образом, перед добавлением нового объекта в массив cs_Links объект класса CSRecords посылает сообщение Equals объекту temp класса CSLink. Если объект с заданным номером студента и номером курса отсутствует в массиве cs_Links, метод AddRecord добавляет объект temp в этот массив, т.е. записывает студента на курс. При этом считается, что номер студента и номер курса заданы верно (существуют). Запись студента на курс завершается успешно, если она не является повторной, и не превышено максимально допустимое количество записей.

Метод Display() распечатывает информацию обо всех объектах массива cs_Links. Так как доступ к полям объектов класса CSLink – частный (private), получить их значения можно только в результате использования метода get соответствующих свойств. Поэтому объект класса CSRecords посылает сообщения каждому объекту массива cs_Links для определения номера студента, номера курса и соответствующей оценки:

```
for (int i=0; i< cur_counts; i++)
  Console.WriteLine("Student:  "
         + cs_Links [i].S_id
         + "Course "
         + cs_Links [i].C_id
         + "  Grade "
         + cs_Links [i].Grade);
```

Описание класса CSRecords представлено в листинге 4.

В результате создания объекта класса CSRecords и записи студентов на курсы создаются и входящие в этот объект компоненты – объекты класса CSLink. Поэтому объект класса CSRecords является составным, а отношение между объектами классов CSRecords и CSLink – отношением включения.

```
class CSRecords {
    CSLink [] cs_Links; //ссылка на массив объектов класса CSLink
    int max_counts; // максимальное количество записей
                    // студентов на курсы
    int cur_counts; // текущее количество записей студентов на курсы
    // описание свойств Max_counts и Cur_counts ...
    public CSRecords(int max_counts) {// конструктор ... }
    //метод проверяет, существует ли запись студента на курс
    public bool CS_Find (int num_st, int num_course){
      int i=0; CSLink temp = new CSLink (num_st, num_course);
      while ((i < cur_counts) && !temp.Equals(cs_Links [i])) i++;
      if (i == cur_counts) return (false); else return (true);
     }
     //метод записывает студента на курс
     public bool AddRecord(int num_st, int num_course){
       if (!CS_Find (num_st, num_course))
        // добавление информации о записи студента
       // на курс  в массив cs_Links   . . .
     }
    // печать информации обо всех записях студентов на курсы
     public void Display(){
        for (int i=0; i< cur_counts; i++)
          Console.WriteLine("Student:  " + cs_Links [i].S_id + "
                            Course " + cs_Links [i].C_id+
                           "  Grade " +cs_Links [i].Grade);
     }
  }
```

Листинг 4. *Описание класса CSRecords*

Еще один класс Recorder содержит список всех студентов и список всех курсов (рис. 7).

Эта информация хранится в массивах list_Student и list_Course соответственно. Массив list_Student может содержать студентов, которые пока не записались ни на один курс; а в массив list_Course могут входить курсы, которые пока никто не выбрал. Кроме того, в классе Recorder описан объект csr_CSRecords класса CSRecords, который связывает студентов, курсы и записи о выбранных студентами курсах.

Методы AddStudent и AddCourse добавляют новых студентов и новые курсы в соответствующие массивы. Сначала метод AddStudent проверяет, существует ли объект с заданным именем и номером студента в массиве list_Student, а именно:

✓ создает объект temp класса Student, содержащий заданные значения имени name и номера студента num_st:

```
Student temp =
     new Student(name, num_st);
```

✓ сравнивает объект temp с объектами массива list_Student, а именно, вызывает метод Equals объекта temp; параметр этого метода – текущий объект массива студентов:

```
int i=0;
while ((i < cur_st) &&
  !temp.Equals(list_Student[i]))
     i++;
```

Таким образом, чтобы избежать повторного включения информации о студенте в массив list_Student, объект класса Recorder посылает сообщение Equals объекту temp класса Student. Если объект с заданным именем и номером студента отсутствует в массиве, объект temp добавляется в массив.

Аналогичная ситуация имеет место и при добавлении курсов: перед включением записи о курсе в массив list_Course объект класса Recorder посылает сообщение Equals объекту temp класса Course.

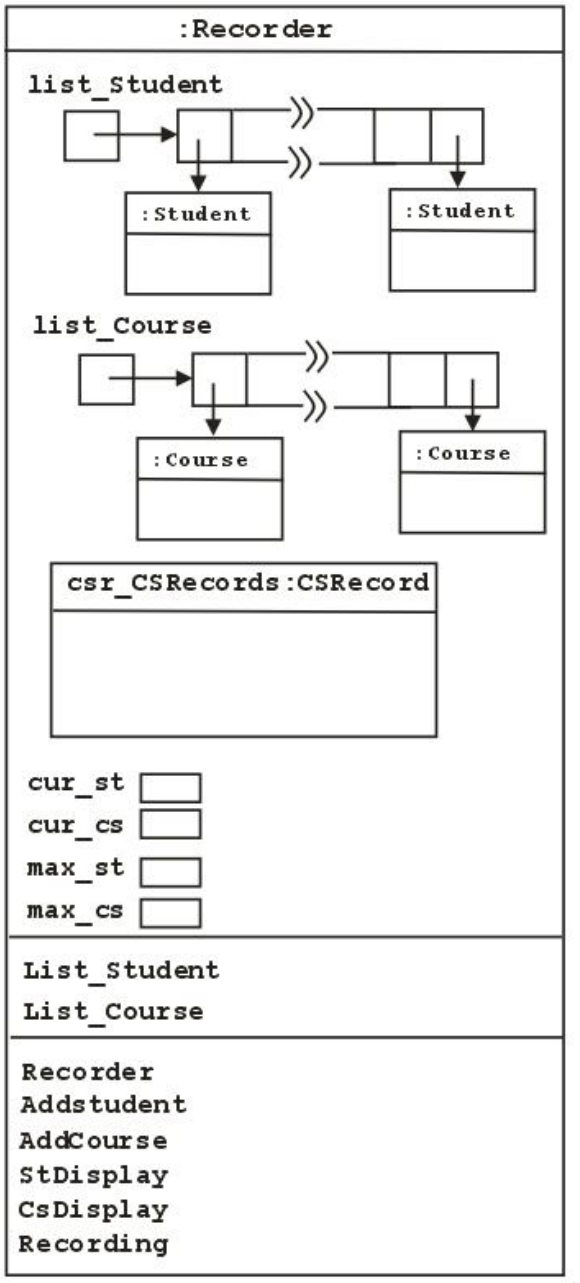


Рис. 7. *Диаграмма объекта класса* Recorder

Как только студенты и курсы добавлены, можно начинать запись студентов на курсы. Для этого используется метод Recording, входные данные которого – номер студента и номер курса. Метод Recording

✓ проверяет наличие заданного студента student_id в массиве list_Student:

```
int i=0;
while ((i < cur_st)&&
       (student_id!=
         list_Student[i].S_id))
  i++;
```

в этом случае объект класса Recorder посылает сообщения каждому объекту массива list_Student для определения номера студента;

✓ проверяет наличие заданного курса course_id в массиве list_Course:

```
int j=0;
while ((j < cur_cs) &&
       (course_id !=
         list_Course[j].C_id))
   j++;
```

здесь объект класса Recorder посылает сообщения каждому объекту массива list_Course;

✓ вызывает метод AddRecord объекта csr_CSRecords, т.е. для записи студента на курс объект класса Recorder посылает сообщение AddRecord объекту csr_CSRecords:

```
csr_CSRecords.AddRecord
        (student_id , course_id).
```

Кроме того, при печати списка студентов и списка курсов объект класса Recorder посылает сообщения объектам класса Student и Course для получения имени и номера студента, а также имени и номера курса (листинг 5).

```
class Recorder {
   Course [] list_Course ; // ссылка на массив курсов
   Student [] list_Student; // ссылка массив студентов
   CSRecords csr_CSRecords; //ссылка на объект класса CSRecords
   int cur_st; //текущее количество студентов
   . . .
   // описание свойств List_Course и List_Student . . .
   // конструктор . . .
   // метод добавляет студента в массив студентов
   public void AddStudent(string name, int num_st) {
    Student temp = new Student(name, num_st); int i = 0;
    //проверяем, есть ли такой студент
    while ((i < cur_st) && !temp.Equals(list_Student[i])) i++;
    if //не нашли,  добавляем объект temp в массив list_Student
      . . .
    }
   public void St_Display(){ //вывод списка студентов
   for (int i = 0; i < cur_st; i++)
     Console.WriteLine(list_Student[i].S_name + "  " +
                                          list_Student[i].S_id);
   }
   // метод добавляет курс в массив курсов
   public void  AddCourse (string name, int num_cs) { . . .  }
   public void Cs_Display(){ //вывод списка курсов . . . }
   // метод записывает студента на курс
   public bool Recording (int student_id, int course_id){
     .  .  .
     if (csr_CSRecords.AddRecord(student_id, course_id))
           return (true);  else return (false);
   }
  }
```

Листинг 5. *Описание класса Recorder*

Таким образом, в процессе использования объекта класса Recorder реализуется взаимодействие объектов классов Student, Course, CSLink и CSRecords.

Важно наглядно представить передачу сообщений между различными объектами.

Для этой цели используются диаграммы взаимодействия и последовательностей [2].

На диаграмме взаимодействия изображаются объекты и их основные связи (рис. 8). Диаграмма же последовательностей описывает последовательность (временной порядок) передачи сообщений (рис. 9).

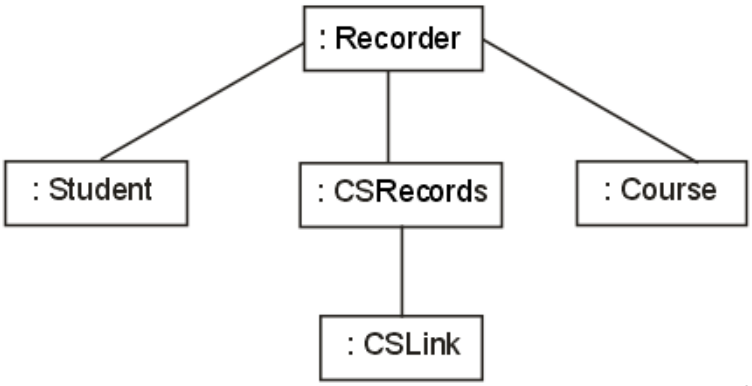


Рис. 8. *Диаграмма взаимодействия для задачи "Курсы по выбору"*

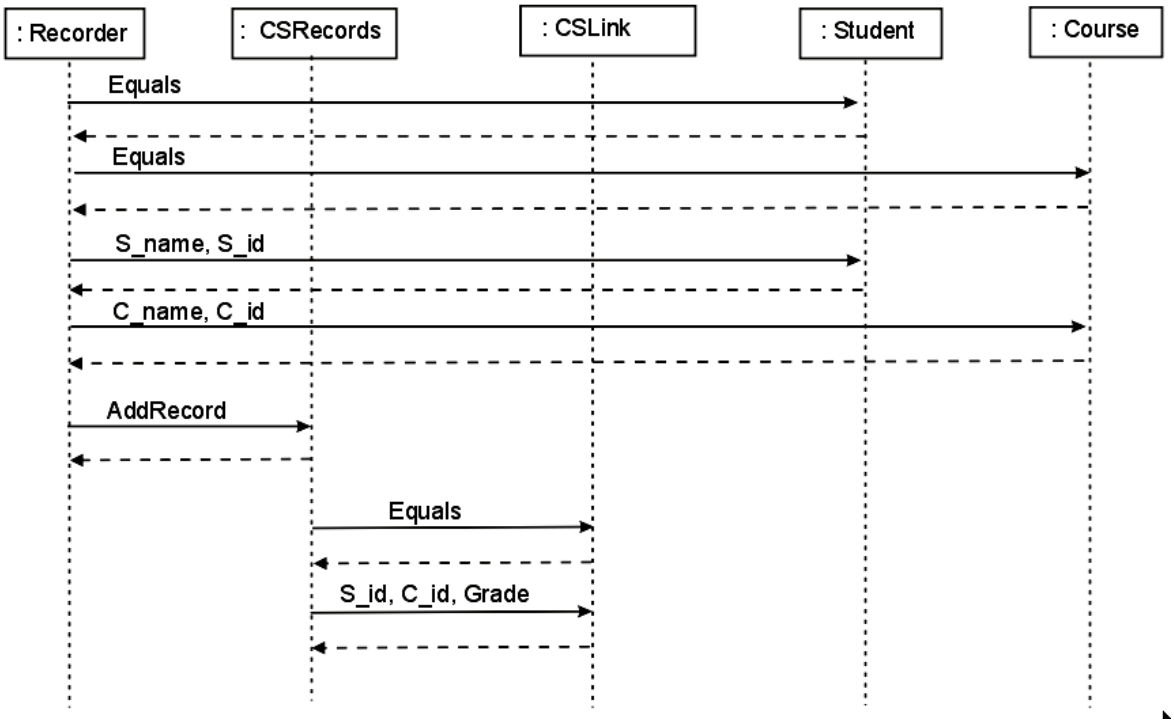


Рис. 9. *Диаграмма последовательностей для задачи "Курсы по выбору"*

Построение диаграммы начинается с размещения по горизонтальной оси объектов, участвующих во взаимодействии. Обычно объекты, инициализирующие взаимодействие, располагаются слева. Вдоль вертикальной оси в хронологическом порядке размещаются сообщения (сверху вниз). Линия жизни объекта – вертикальная пунктирная линия – показывает продолжительность существования объекта. Большинство объектов существуют в течение всего периода взаимодействия, поэтому их линии жизни выровнены по верхней границе диаграммы и имеют одинаковую длину. Сообщения на диаграмме изображаются стрелками, направленными от одной линии жизни к другой. Сплошная стрелка указывает на объект, который принимает сообщение, а пунктирная – на объект, которому возвращается результат.

В статье представлено частичное решение задачи "Курсы по выбору", которое отвечает лишь за запись студентов на курсы. В дальнейшем это решение следует дополнить реализацией следующих действий:

- выставить оценку студенту по курсу;
- выяснить, сколько студентов не сдали курс;
- установить, какое количество студентов успешно освоили курс;
- выявить, на какой курс записалось наибольшее количество студентов

и др.

Список литературы

1. *Гради Буч и др.* Объектно-ориентированный анализ и проектирование с примерами приложений. М.: Вильямс, 2010.
2. *Гради Буч и др.* Язык UML. Руководство пользователя. М.: ДМК Пресс, 2007
3. *Герберт Шилдт.* Полный справочник по C#. М.: Вильямс, 2007.

Object interaction in object-oriented programming

L. A. Zalogova
Perm State University; 15, Bukireva st., Perm, 614990, Russia
zalogova.la@gmail.ru

An object-oriented program consists of a set of objects. The usage of isolated objects solves only a small number of tasks. Typically, they are the objects which exchange messages, i.e. interact with each other, that are of interest. The paper specifies one of the possible ways to organize object interaction and provides fragments of a programming code for a specific example. To visualize the message exchange between objects, interaction diagrams and sequence diagrams are described. The proposed methods can be used in teaching the object-oriented programming technology.

Keywords: *object-oriented programming; class; object; object interaction.*

Object interaction in object-oriented programming

L. A. Zalogova
Perm State University; 15, Bukireva st., Perm, 614990, Russia
zalogova.la@gmail.ru

Object-oriented program consists of a set of objects. The usage of the isolated objects solves only a small number of tasks. Typically the objects, which exchange the messages, that is interact with each other, are of great interest. The paper specifies one of the possible ways to organize the object interaction; the fragments of programming code are given for a specific example. Interaction diagrams and sequence diagrams are described to visualize the message exchange between the objects. The suggested methodology can be used in object-oriented programming technology teaching.

Keywords: *object-oriented programming; class; object; object interaction.*