Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

ОО проект:
Компьютерная модель высотного лифта

Проф. Забудский Е.И.

Москва 2008

Тема 9. Разработка компьютерных моделей
на основе объектно-ориентированной методологии:
практический пример

ОО проект:
Компьютерная модель высотного лифта
(консольный вариант).

GUI выполняется студентами самостоятельно
на основе шаблона Модель – Вид – Контроллер (см. лекция 15, с. 4…23)

Unified Modeling Language (UML) – популярный язык графического моделирования, используемый для представления объектно-ориентированных программ (**www.omg.org/uml**).

// **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

Моделирование высотного лифта

Моделирование высотного лифта

В примере с помощью классов C# моделируется система лифтов.

1. Из чего должна состоять система Компьютерная модель высотного лифта?

1. Обычно в большом здании располагается несколько одинаковых лифтов.
2. На каждом этаже есть кнопки «**Вниз - Down**» и «**Вверх - Up**».
3. В большинстве случаев на этаж выделяется одна пара таких кнопок.
4. Когда вы вызываете лифт, вы не знаете, какой именно из них приедет раньше.
5. Внутри лифта кнопочек существенно больше — одна для каждого этажа.
6. Войдя в лифт, пассажиры обычно нажимают кнопку нужного этажа.

Программа моделирует все эти составляющие процесса.

2. Работа программы ELEV

При запуске вы увидите четыре лифта, находящихся внизу экрана, а также вертикальный список чисел слева от **1** до **20**, увеличивающихся снизу вверх **(см. стр. 23…30 на с. 13)**. Изначально все лифты на первом этаже. Лифт обозначен тремя символами **WWW**. Это все показано на рис. 1

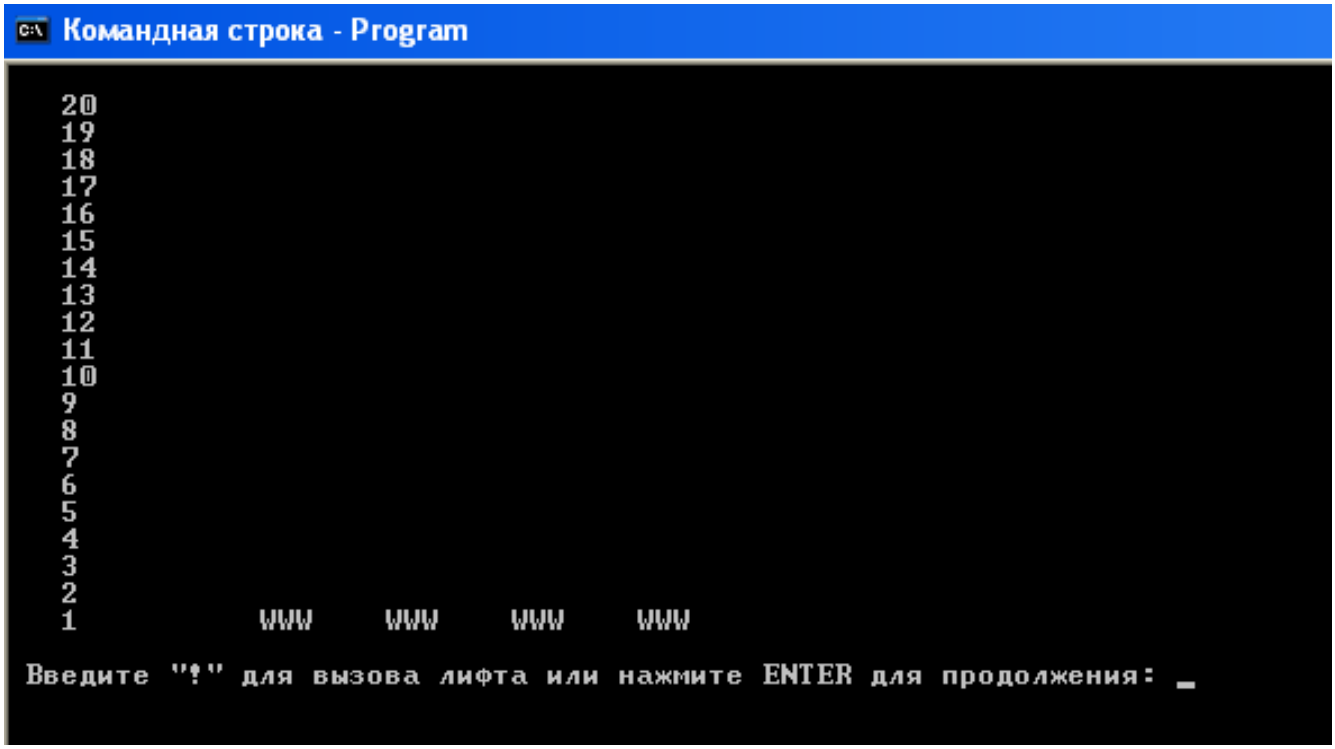


Рис. 1. Начальное состояние программы **ELEV**

2.1. Запрос этажа

При вводе символа «**!**» и нажатии **Enter** внизу экрана появится текст **(см. стр. 102 на с. 14)**:

На каком Вы этаже? **(см. стр. 109 на с. 14)**

Можно ввести любое число от **1** до **20**.

Если вы только что приехали на работу и намерены подняться к своему рабочему месту, нажимайте **1**. Если спешите спуститься на обед со своего этажа, введите его номер.

Следующее, о чем спросит программа, это

2.2. В каком направлении будете двигаться (**u** / **d**) **(см. стр. 111 на с. 14)**:

Подсказка: если вы на первом этаже, вам, скорее всего, надо вверх (**u**);

если на последнем — вниз (**d**).

После ввода « **u** » отображается символ « **^** ». После ввода « **d** » – символ « **v** ».

В качестве домашнего задания продумайте, куда можно ехать с какого-то из средних этажей. Когда вы разберетесь с тем, где вы находитесь и куда вам нужно, напротив этажа назначения появится символ звездочка « * ». По мере возникновения других запросов звездочка будет появляться рядом с номерами запрашиваемых этажей.

Если лифт находится там же, где и вы, его двери немедленно откроются. Что касается программы, то перед вашими глазами возникнет счастливое лицо (☺) офисного работника, заходящего вовнутрь. В противном случае лифт начнет двигаться вверх или вниз (это происходит при нажатии на “Enter”), пока не достигнет этажа, с которого был сделан запрос.

2.3. Ввод конечного пункта путешествия / этаж назначения /(см. стр. 303 на с. 20)

Пока счастливый пассажир находится внутри лифта, нужно быстренько поинтересоваться у него, куда это он собрался ехать:

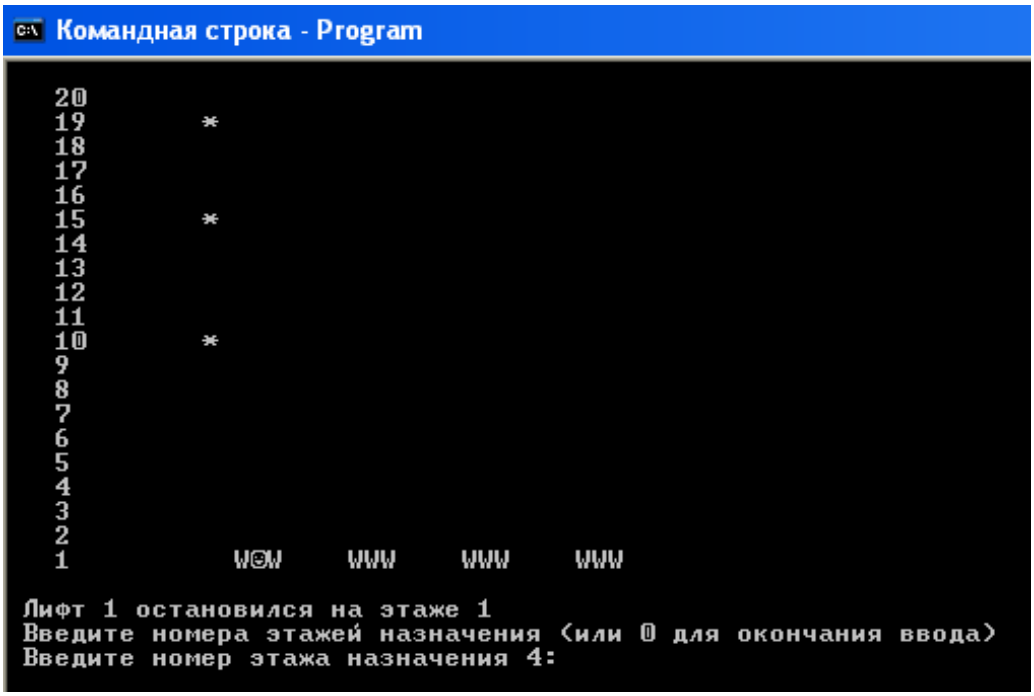
Лифт 1 находится на этаже 1.
Введите номера этажей назначения (или **0** для окончания ввода)
Этаж назначения 1-й: 13

Пассажир ввел **13.** Но пока он размышлял, куда он хочет прокатиться, пришли еще пассажиры и стали нажимать разные кнопочки в лифте, всем ведь нужно на разные этажи! Поэтому программа должна позволять ввести несколько номеров этажей. Введите несколько номеров (не больше **20**), затем обязательно нажмите **0** для окончания ввода.

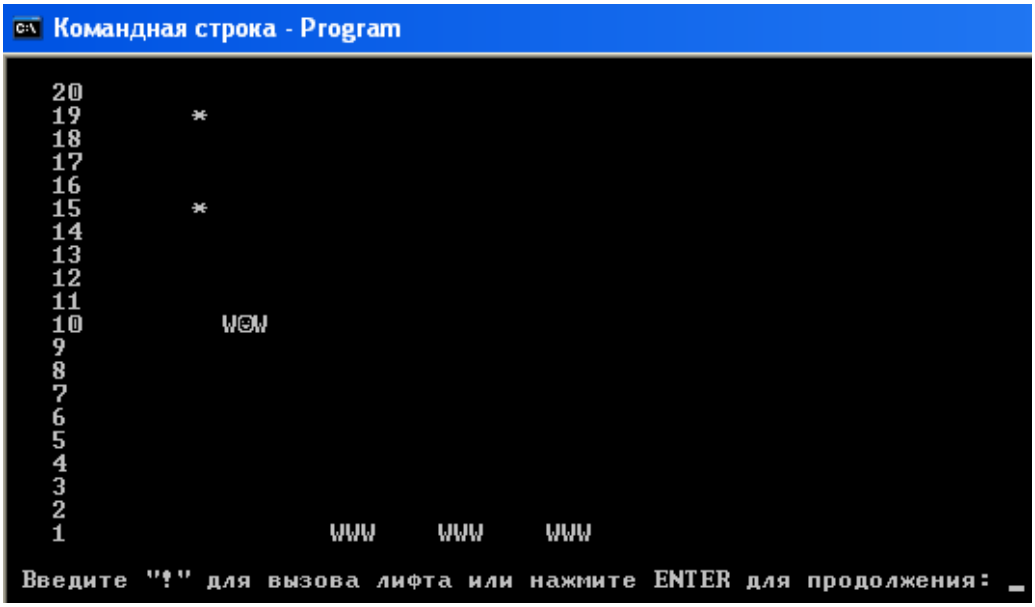
Пункты назначения, указанные пассажирами, отображаются символом звездочка « * » **(см. стр. 214 на с. 18)**.

Каждый лифт имеет свой набор пунктов назначения (в отличие от запросов с исходных этажей, общих для всех лифтов).

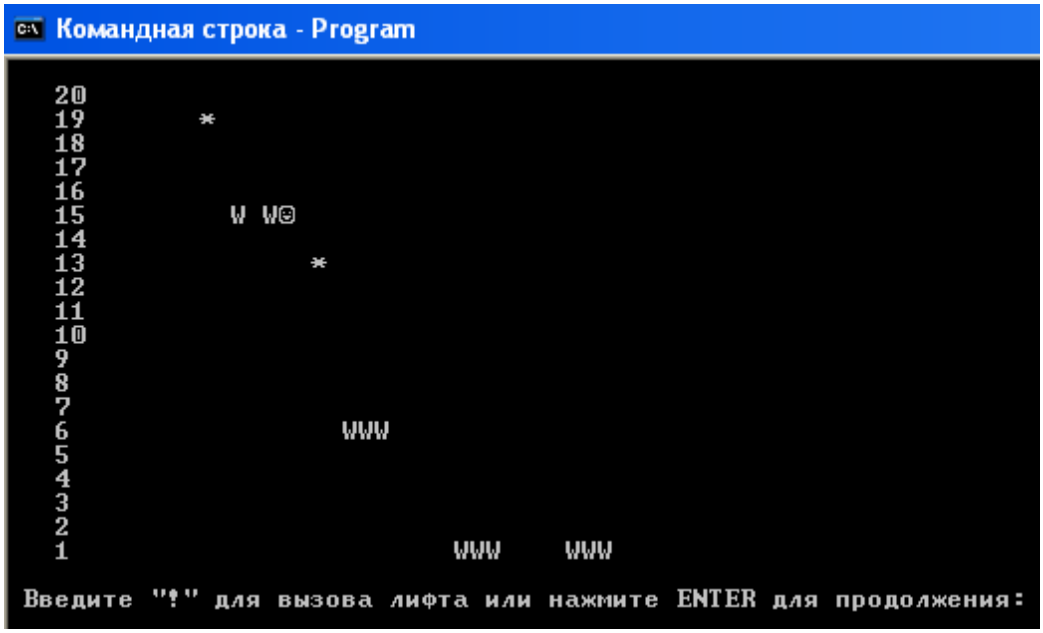
Запросов с разных этажей можно делать сколько угодно. Программа запомнит их, как и выбранные из каждого лифта этажи назначения, и будет пытаться обслужить всех. Все четыре лифта могут быть одновременно в движении. На рис. 2*а* показана ситуация с одним запросом с 1-го этажа и несколькими запросами пунктов назначения: этажи 10, 15 и 19-й.



а



б



в

Рис. 2. Лифты в действии:
а – три пассажира в 1-м лифте W☺W на 1-м этаже;
б – один пассажир переместился в 1-м лифте W☺W на 10-й этаж;
в – другой пассажир переместился на 15-й этаж и выходит из 1-го лифта WW☺,
одновременно 2-й лифт с одним пассажиром перемещается на 13-й этаж;

3. Проектирование системы

3.1. Система лифтов (класс Elevator, см. Листинг на с. 14…20)

Лифты могут быть приблизительно одинаковыми, поэтому есть смысл сделать их объектами одного класса под названием **Elevator**. В этом классе содержатся данные, характерные для каждого отдельно взятого лифта: текущее местоположение, направление движения, номера этажей назначения и т. д.

Методы **Elevator** инициализируют с помощью конструктора отдельные лифты, устанавливают два метронома для каждой кабинки, выводят изображение лифтов, их направления, принимают решения, двигают лифты и получают номера конечных этажей от пользователя.

## строк	Elevator	Класс **Elevator, см. Листинг на с. 14…20**
9…16	-car_number	номер лифта
-“-	-current_dir	в каком направлении едем?
-“-	-current_floor	текущий этаж: где мы?
-“-	-destination	направление – выбирается пассажиром
-“-	-old_floor	куда едем?
-“-	-ptrBuilding	
-“-	-loading_timer	таймер (не нуль) – во время посадки
-“-	-unloading_timer	таймер (не нуль) – во время высадки
18..	+Elevator	конструктор
220..	+car_display	вывод образа лифта
34..	+car_tick1	метроном-1 для каждого лифта
270..	+car_tick2	метроном-2 для каждого лифта
45..	+decide	принятие решения
208..	+dests_display	вывести конечные этажи, выбраные кнопками
292..	+get_direction	получение текущего направления
297..	+get_distinations	получить конечный этаж
286..	+get_floor	получение текущего этажа
275	+ move	движение лифта

Рис. 3. Диаграмма класса **Elevator**

3.2. Здание (класс Building, см. Листинг на с. 12…14)

Есть данные обо всем здании в целом. Эти данные будут частью класса **Building**. Во-первых, есть массив *запросов с этажей* **floor_request** **(см. стр. 9 на с. 12)**. Это список этажей, где люди, ждущие лифта, нажали кнопку «вверх» или «вниз» и тем самым попросили лифт подъехать к ним. Любой из лифтов может отвечать на эти запросы, поэтому все они должны иметь доступ к этому массиву. В программе используется массив размером ***N*x2** типа **booL**, где ***N*** — число этажей, а два (**2**) поля на каждый этаж позволяют разделять запросы на движение **вверх** и **вниз** **(см. стр. 17 на с. 12)**. Все лифты обращаются к этому массиву, чтобы понять, что им делать дальше.

Кроме того, каждый лифт должен быть осведомлен о том, где находятся остальные. Если один из лифтов находится на первом этаже, то нет никакого резона гнать его на пятнадцатый, если есть лифт, находящийся на десятом. Выполнять запросы должен ближайший к месту назначения лифт. Для обеспечения осведомления лифтов обо всех других в класс **Building** вводится также массив **car_list** **(см. стр. 10 на с. 12)**. Массив **car_list** позволяет каждой кабинке опрашивать все другие, выясняя их местонахождение и направление движения.

Методы **Building** инициализируют систему, устанавливают метроном, получают и выводят запросы.

## строк	Building	Класс **Building см. Листинг на с. 12…14**
9, 10	-car_list	массив лифтов
-“-	-floor_request	массив запросов с этажей
13..	+Building	конструктор
74..	+ get_cars_dir	выяснение направления движения
69..	+ get_cars_floor	поиск лифта
64..	+ set_floor_req	установка запросов с этажа
49..	+ get_floor_req	проверка запросов с этажа
33..	+ master_tick	метроном – рассылка временных меток всем лифтам
96..	+ record_floor_reqs	получение запросов с этажа
79..	+ show_floor_reqs	вывод запросов

Рис. 4. Диаграмма класса **Building**

4. Управление временем

В методе **Main()** один из методов класса **Building** вызывается через определенные интервалы времени, чтобы моделирование было более динамичным. Этот метод называется **master_tick() (см. на с.13)**. Он, в свою очередь, вызывает функцию для каждого лифта под названием **car_tickl()**. Она, кроме всего прочего, прорисовывает лифт на экране и вызывает еще одну функцию для принятия решения о том, что делать дальше. Выбор действий богат: **1)** поехать вниз, **2)** поехать вверх, **3)** остановиться, **4)** посадить пассажира, **5)** выпустить пассажира.

После этого каждый лифт должен быть сдвинут на свою новую позицию. Так. Что-то в этом месте все усложняется. Поскольку каждый лифт должен знать, где находятся остальные, прежде чем принять решение, все лифты должны пройти через процесс принятия решения, прежде чем кто-либо из них сдвинется с места. Чтобы удостовериться в том, что это действительно имеет место, мы запускаем **tick'и** дважды для каждой кабинки. Таким образом, **car_tick1() (см. на с.15)** вызывается для принятия решения о том, куда каждый лифт поедет, а другая функция — **car_tick2() (см. на с.19)**— для реального запуска лифтов. Изменяется значение переменной **current_floor** — кабинки начинают двигаться.

Процесс **посадки** пассажиров включает в себя последовательность определенных шагов, которые на экране отображаются так **(см. на с.19)** :

1. кабинка с закрытыми дверями, счастливое лицо отсутствует;
2. кабинка с открытыми дверями, счастливое лицо слева ☺ **(стр. 228 на с. 19)**;
3. кабинка со счастливым лицом ☺ в открытых дверях, узнать от пассажира конечный этаж **(стр. 231, 232 на с. 19)**;
4. кабинка с закрытыми дверями, счастливое лицо отсутствует.

При **высадке** пассажира применяется обратный порядок **(см. на с.19)**. Эти последовательности действий осуществляются с помощью таймера (целочисленной переменной), который уменьшается с каждой временной отметкой (**tick**) от **3** до **0**. С помощью **case** в функции **car_display() (см. на с. 18, 19)** выбирается и рисуется на экране соответствующая версия изображения лифта.

5. Листинг программы ELEV

Программа содержит три класса: **Building (с. 12…14), Elevator (с. 14…20) и Program** с методом **Main (с. 12)**, а также перечисление **Direction (см. строку 7 на с. 12 /верх/).**

6. Стратегия работы лифтов

Встраивание интеллекта необходимого уровня в лифтовые системы — не простая задача. Процесс принятия решения отрабатывается функцией **decide() (см. на с. 15…18)**, состоящей из определенного набора правил. Эти правила организованы в порядке их приоритета. Если применяется одно из них, то выполняется некое действие, при этом правила более низких уровней не отрабатываются. Приведем упрощенный вариант такой обработки:

1. Если лифт вот-вот врежется в дно шахты или пробьет ее крышу, наверное, следует остановиться.
2. Если данный этаж — это этаж назначения, высадить пассажиров.
3. Если обнаружен на данном этаже запрос «вверх», едем вверх, загружаем пассажиров.
4. Если обнаружен на данном этаже запрос «вниз», едем вниз, загружаем пассажиров.
5. Если нет запросов этажей или этажей назначения ни снизу, ни сверху, остановиться.
6. Если этажи назначения сверху, едем вверх.
7. Если этажи назначения снизу, едем вниз.
8. Если стоим или движемся вверх, есть запрос с более высокого этажа, и нет при этом лифтов, движущихся вверх, между нами и этажом запроса или над ним, а также движущихся вниз и находящихся ближе к нему, чем мы, то едем вверх.
9. Если стоим или движемся вниз, есть запрос с более низкого этажа, и нет при этом лифтов, движущихся вниз, между нами и этажом запроса или под ним, а также движущихся вверх и находящихся ближе к нему, чем мы, то едем вниз.
10. Если ничего никому от нас не нужно, останавливаемся.

Правила **8** и **9** довольно сложны. Они предназначены для исключения выполнения одного и того же запроса несколькими лифтами сразу. Тем не менее результаты не всегда идеальны. В некоторых ситуациях лифты медлят с принятием решения, опасаясь сделать то, что могут сделать за них другие. Но в реальности в этот момент другие лифты не движутся к той же цели, а, например, отвечают на свои запросы. Чтобы улучшить стратегию работы системы, необходимо заставить различать функцию **decide()** запросы «вверх» и «вниз» во время проверки относительного местонахождения **ЗЭ** (запроса с этажа). Но это еще больше усложнило бы и без того слишком длинную функцию.

7. Диаграмма состояний для программы ELEV

Рассмотрим диаграмму состояний для программы моделирования лифтов. С целью упрощения задачи, будем считать, что в здании находится только **один человек** и имеется только **один лифт**. Таким образом, в один момент времени может быть только один запрос с этажа и один этаж назначения. Лифту не нужно следить за другими. Диаграмма состояний представлена на рис. 5.

Обозначения в диаграмме таковы:

«**cd**» — этаж назначения, то есть кнопка, нажатая внутри лифта (грубо говоря, это соответствует массиву **destination** из программы);

«**fr**» — запрос с этажа, кнопка, нажатая снаружи лифта (грубо говоря, это значение переменной **floor_req).**

Состояния лифта определяются по значениям переменной **current_dir** (в каком направлении едем? **(см. на с. 17)** и состояниям **loading_timer** (посадка **|см. на с. 18, 19|**) и **unloading_timer** (высадка **|см. на с. 19|**). Поскольку все состояния переходят друг в друга с помощью слова «**вдруг**» (**то есть по сигналам таймера**), на диаграмме показаны только контрольные состояния. Отображаются представления лифта: **1)** о запросах с этажей («**fr**») и **2)** о запросах этажей назначения («**cd**»).

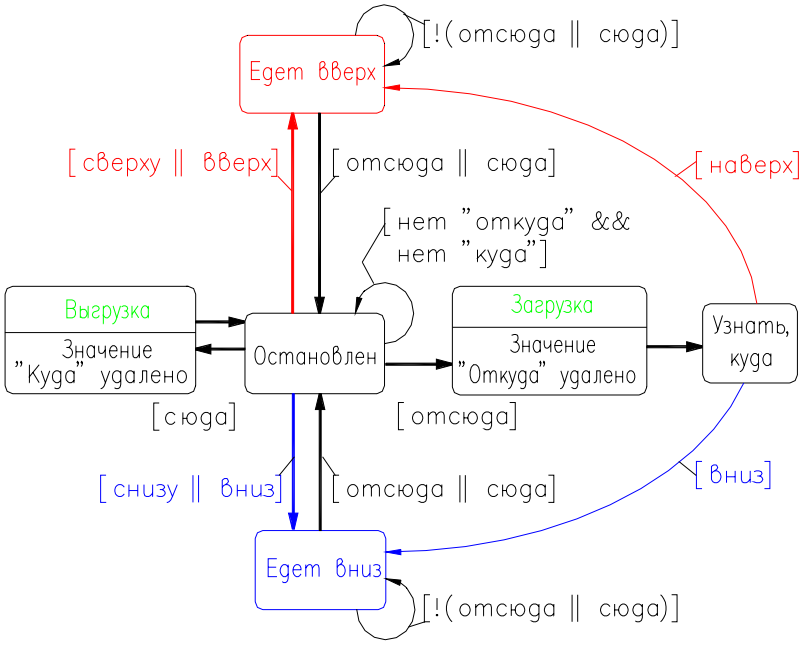


Рис. 5. Диаграмма состояний объекта Elevator

7.1. Особенности диаграмм состояний

На **диаграммах классов** UML отражаются взаимоотношения между классами. В диаграмме классов отражена организация кода программы. Диаграммы классов – это **статические** диаграммы, в которых связи не изменяются при запуске программы.

Но иногда полезно рассмотреть объекты классов в **динамическом** режиме. С момента своего создания объект вовлекается в деятельность программы, выполняет различные действия и в конечном итоге удаляется. Ситуация постоянно изменяется, и это графически отражено на диаграмме состояний.

Диаграммы состояний UML показывают, как изменяются с течением времени ситуации, в которых находится объект. Состояния показываются на диаграммах в виде прямоугольников со скругленными углами, а переходы между состояниями — в виде прямых линий.

Между состояниями существуют переходы. Получив сигнал от пульта управления, телевизор переключается из активного состояния канала 7 в активное состояние канала 2.

На рис. 5 показана диаграмма состояния для программы **ELEV**. На ней отражены различные состояния объекта класса **Elevator**, которые он может принимать во время работы программы.

7.1.1. Состояния

В диаграммах состояний UML состояние представлено в виде прямоугольника со скругленными углами. Название состояния указано в верхней части прямоугольника, обычно оно начинается с заглавной буквы. Ниже указаны действия, которые выполняет объект, входя в это состояние.

После создания объект класса **Elevator** может пребывать в пяти состояниях: Загрузка, Выгрузка, Едет вверх, Едет вниз и Остановлен.

В отличие от диаграмм классов, в коде программы нельзя точно указать фрагмент, соответствующий каждому конкретному состоянию. Чтобы разобраться в том, какие состояния должны входить в диаграмму, нужно представить ситуацию, в которой работает объект, и то, что мы хотим получить в результате. Затем каждому состоянию подбирают подходящее название.

7.1.2. Переходы

Переходы между состояниями представлены в диаграммах в виде стрелок, направленных от одного прямоугольника к другому. Если переход обусловлен каким-то событием, то он может быть обозначен его именем. Имена могут быть более приближены к реальному языку, чем к терминам C#.

Переходы могут быть также отмечены тем, что в **UML** называют защитой: это условие, которое должно быть выполнено для совершения перехода. Оно записывается в квадратных скобках. Переходы имеют защиту и имя события.

Заметим, что три перехода является переходами сами в себя, они возвращает нас в то же состояние.

7.1.3. От состояния к состоянию

Каждый раз, попадая в состояние Загрузка, объект класса Elevator выполняет действие, заключающееся в удалении значения «Откуда». Попадая в состояние Выгрузка, объект класса Elevator выполняет действие, заключающееся в удалении значения «Куда».

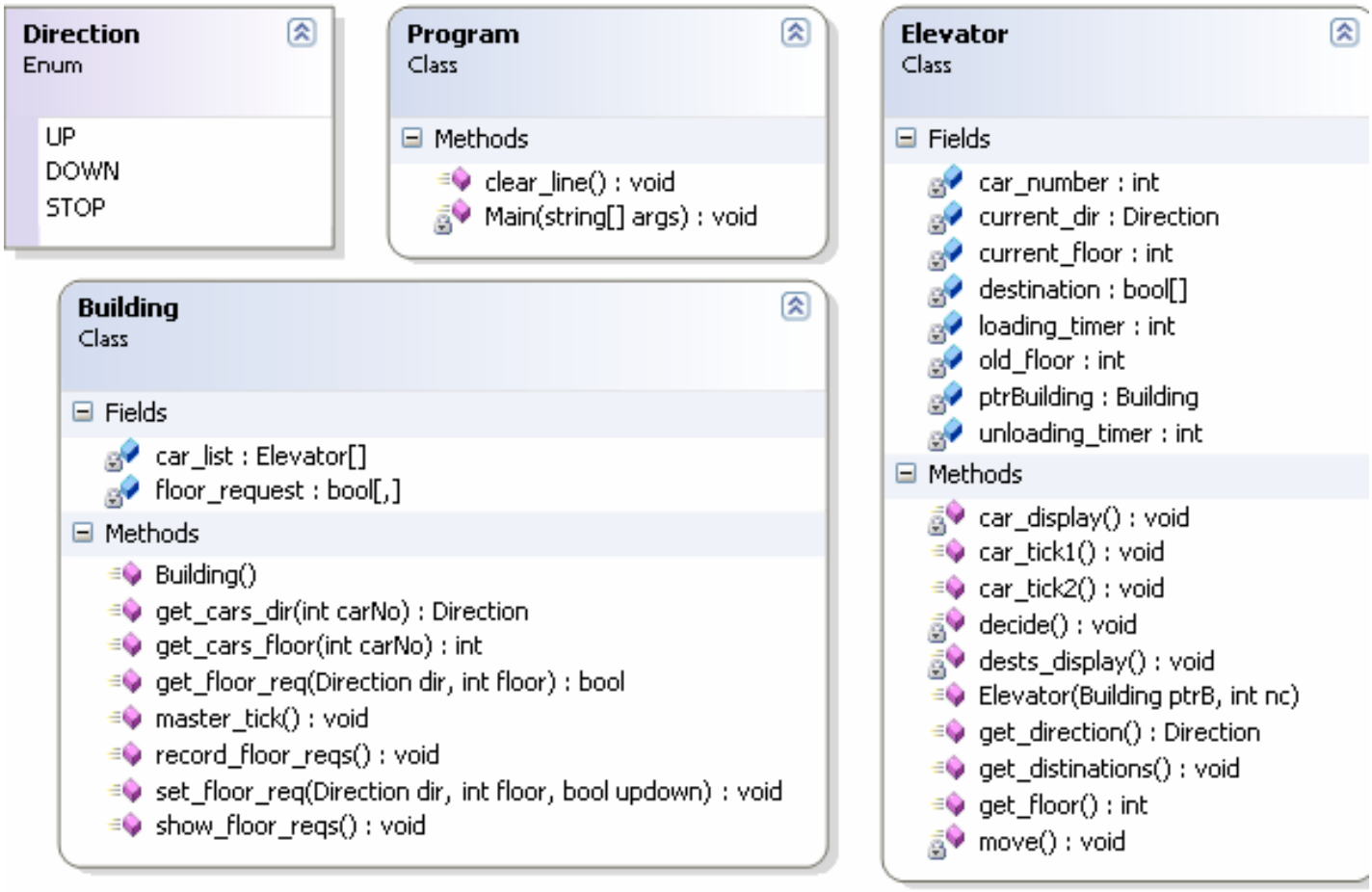


Рис. 6. Диаграмма классов программы ELEV

Задание студентам

1. Разберитесь в коде программы **ELEV. Для анализа** перепишите программу **ELEV** таким образом, чтобы она работала только с одним лифтом. Это сильно упростит программное решение. Удалите ненужные части программы. Кроме того, можно представить себе задачу с одним лифтом и одним пассажиром, что еще больше упростит дело.

2. Модифицируйте программу работы с лифтами, улучшив методы обработки запросов. Чтобы увидеть огрехи в поведении нынешнего варианта программы, попробуйте создать запрос «вниз» с двадцатого этажа. Затем создайте запрос «вниз» с десятого этажа. **Лифт-1** тотчас поднимется на **20** этаж. Но **лифт-2**, который, по идее, должен подниматься на **10** этаж, будет ждать, пока **лифт-1** пройдет **10** этаж, только после этого начнет движение. Измените функцию **decide()** таким образом, чтобы исключить эту ошибку.

3. Реализуйте WINDOWS-интерфейс программы **ELEV – проектируйте: лифты должны перемещаться, при этом номер текущего этажа должен изменяться – инкрементироваться или декрементироваться; двери закрываться и открываться; пассвжиры входить и выходить из лифта. Осуществите вывод статистики: сколько этажей прошел каждый лифт за сеанс моделирования; сколько пассажиров перевез каждый лифт? Думайте. Ваши предложения?!!**

4. Создайте библиотеку классов, моделирующую какую-нибудь интересующую вас предметную область. Создайте **Main()** или клиентскую программу для ее тестирования. Предложите свою библиотеку классов на рынке, станьте богатым и знаменитым.

```
1:  using System;
2:  using System.Collections.Generic;
3:  using System.Text;
4:
5:  namespace ELEV
6:  {                          //перечисление  Direction – направление движения
7:     public enum Direction { UP = 0, DOWN = 1, STOP = 2}  // список констант
8:
9:     class Program                              //класс Program
10:    {
11:       static void Main(string[ ] args)
12:       {
13:          Building theBuilding = new Building();  //создаем здание. С. 13…31 кл. Building
14:          Console.SetCursorPosition(1, 24);
15:          Console.WriteLine();  //переводим на строку вниз, чтобы потом не скакать
16:          while (true)
17:          {
18:             theBuilding.master_tick();
19:             //System.Threading.Thread.Sleep(1000);          //таймер
20:             theBuilding.record_floor_reqs();                    // с. 96…122
21:          }
22:       }
23:
24:       public static void clear_line()                    //метод очистки строки
25:       {
26:          for (int i = 0; i < 79; i++)  Console.Write(" ");
27:       }
28:    }
29: }                             //окончание класса Program
```

```
1:  using System;                              // класс Building, с. 12…14
2:  using System.Collections.Generic;
3:  using System.Text;
4:
5:  namespace ELEV
6:  {
7:     public class Building
8:     {
9:        private bool[,] floor_request;                  //массив запросов с этажей
10:       Elevator[] car_list;                  // кабинки - лифты
11:
12:                                 //конструктор
13:       public Building()
14:       {
15:          Console.Clear();                //очистка экрана
16:          car_list = new Elevator[ 4 ];                // объявить массив из 4-х лифтов
17:          floor_request = new bool[ 2, 20 ]; // 2 – два направления (UP, DOWN) ; 20 этажей
18:          for (int i = 0; i < 4; i++)                    //создать лифты
19:          {
20:             car_list[i] = new Elevator(this, i);                 // С. 18…32 кл. Elevator
21:          }
```

```
22:
23:             for (int i = 0; i < 20; i++)                        //для каждого этажа
24:             {
25:                 Console.SetCursorPosition(3, 20 - i);           //вывести номер этажа на экран
26:                 Console.Write((i + 1));
27:                 if (i < 10) Console.Write(" ");
28:                 floor_request[(int)Direction.UP, i] = false;    //запросов еще не было
29:                 floor_request[(int)Direction.DOWN, i] = false;
30:             }
31:         }
32:
33:         public void master_tick()               //  метроном
34:         {
35:             int j;
36:             show_floor_reqs();                  //вывести запросы с этажей
37:             for (j = 0; j < 4; j++)                                 //для каждого лифта
38:             {
39:                 Elevator el = (Elevator)car_list[j];
40:         el.car_tick1();  //послать временную метку 1 (с. 34…43). Ответ на вопрос: куда каждый лифт поедет?
41:             }
42:             for (j = 0; j < 4; j++)
43:             {
44:                 Elevator el = (Elevator)car_list[j];
45:          el.car_tick2();  //послать временную метку 2 (с. 270…273). Реальный запуск каждого лифта
46:             }
47:         }
48:
49:         public bool get_floor_req(Direction dir, int floor)       //проверка наличия запросов
50:         {
61:             return floor_request[(int)dir, floor];
62:         }
63:
64:         public void set_floor_req(Direction dir, int floor, bool updown)  //установить запрос
65:         {
66:             floor_request[(int)dir, floor] = updown;
67:         }
68:
69:         public int get_cars_floor(int carNo)          //найти кабинку
70:         {
71:             return car_list[carNo].get_floor();
72:         }
73:
74:         public Direction get_cars_dir(int carNo)          //определение направления
75:         {
76:             return car_list[carNo].get_direction();
77:         }
78:
79:         public void show_floor_reqs()                 //вывод запросов
80:         {
81:             for (int j = 0; j < 20; j++)
82:             {
83:                 Console.SetCursorPosition(7, 20 - j);
84:
```

```
85:             if (floor_request[(int)Direction.UP, j])      Console.Write("^");      //стрелка вверх
86:             else                                          Console.Write(" ");
87:
88:             Console.SetCursorPosition(7 + 3, 20 - j);
89:
90:
91:             if (floor_request[(int)Direction.DOWN, j]) Console.Write("v");      //стрелка вниз
92:             else                                          Console.Write(" ");
93:         }
94:     }
95:
96:     public void record_floor_reqs()         //получение запросов с этажа
97:     {           // эквивалентно нажатию кнопок вызова лифта на этаже
98:         char[] ustring = new char[80];
99:         int iFloor;
100:        char chDirection;
101:        Console.SetCursorPosition(1, 22);                  //низ экрана
102:  Console.Write("Введите \"!\" для вызова лифта или нажмите ENTER для продолжения: ");
103:        string en = Console.ReadLine();                   //ожидание ввода
104:        if (en != "!")   return;                  //если ввели не "!", то ввод отменяется
105:
106:        Console.SetCursorPosition(1, 22);
107:        Program.clear_line();
108:        Console.SetCursorPosition(1, 22);
109:        Console.Write("На каком вы этаже? ");
110:        iFloor = int.Parse(Console.ReadLine());                  //получить этаж
111:        Console.Write(" В каком направлении будете двигаться (u/d): ");
112:        string dir = Console.ReadLine();                 //получить направление
113:        chDirection = dir[0];                        //получаем символ
114:  if (chDirection == 'u') floor_request[(int)Direction.UP, iFloor - 1] = true;  //запрос "вверх"
115:  if (chDirection == 'd') floor_request[(int)Direction.DOWN, iFloor - 1] = true; //запрос "вниз"
116:        Console.SetCursorPosition(1, 22);                  //стереть старый текст
117:        Program.clear_line();
118:        Console.SetCursorPosition(1, 23);
119:        Program.clear_line();
120:        Console.SetCursorPosition(1, 24);
121:        Program.clear_line();
122:    }
123:  }
124: }                               //окончание класса Building
```

```
1:  using System;                          // класс Elevator, с. 14…20
2:  using System.Collections.Generic;
3:  using System.Text;
4:
5:  namespace ELEV
6:  {
7:      public class Elevator
8:      {
9:          private Building ptrBuilding;
10:         private int car_number;
```

```
11:         private int current_floor;
12:         private int old_floor;
13:         private Direction current_dir;
14:         private bool[] destination;
15:         private int loading_timer;
16:         private int unloading_timer;
17:
18:         public Elevator(Building ptrB, int nc)                    //конструктор
19:         {
20:             ptrBuilding = ptrB;
21:             car_number = nc;
22:             current_floor = 0;                                    //начать с 0 (для пользователя - 1)
23:             old_floor = 0;                                        //запомнить предыдущий этаж
24:             destination = new bool[20];
25:             current_dir = Direction.STOP;                         //вначале стоит на месте
26:             for (int j = 0; j < 20; j++)
27:             {
28:                 destination[j] = false;                           //пассажиры еще не нажимали кнопки
29:             }
30:             loading_timer = 0;                                    //еще не началась посадка
31:             unloading_timer = 0;                                  //еще не началась высадка
32:         }
33:
34:         public void car_tick1()                                   //метроном-1 для каждого лифта
35:         {
36:             car_display();                  //нарисовать кабинку, с. 220…268
37:             dests_display();                //нарисовать конечные этажи (назначения), с. 208…218
38:
39:             if (loading_timer > 0)        --loading_timer;                //счет времени посадки
40:             if (unloading_timer > 0)      --unloading_timer;              //счет времени высадки
41:
42:             decide();                                             //принятие решения
43:         }
44:
45:         private void decide()                   //принятие решения    (строки 45…206)
46:         {
47:             int j;
48:                                         //флаги показывают, сверху или снизу запросы или назначения
49:             bool destins_above, destins_below;                            //конечные пункты
50:             bool requests_above = false, requests_below = false;          //запросы
61:             int nearlest_higher_req = 0;
62:             int nearlest_lower_req = 0;
63:         /*  флаги указывают, есть ли другие лифты, двигающиеся в том же направлении между
                                                                  нами и ближайшим запросом с этажа*/
64:             bool car_between_up, car_between_dn;
65:             bool car_opposite_up, car_opposite_dn;
66:             int ofloor;                                           //этаж
67:             Direction odir;                                       //направление
68:
69:                                                 //убедиться, что мы не слишком высоко или низко
70:     if ((current_floor == 20 - 1 && current_dir == Direction.UP) ||
        (current_floor == 0 && current_dir == Direction.DOWN))     current_dir = Direction.STOP;
71:
```

```
72:
73:                                                    //если этаж назначения - текущий, начать высадку
74:             if (destination[current_floor] == true)
75:             {
76:                 destination[current_floor] = false;       //удалить это назначение
77:                 if (unloading_timer == 0)                 //высадка
78:                     unloading_timer = 3;
79:                 return;
80:             }
81:
82:                         /*если есть запрос "вверх" с этого этажа и если мы едем вверх или стоим,
                                                                произвести посадку на борт*/
83:     if (ptrBuilding.get_floor_req(Direction.UP, current_floor) && current_dir != Direction.DOWN)
84:             {
85:                 current_dir = Direction.UP;                 //(если была остановка)
86:                                                    //удалить ЗЭ в данном направлении движения
87:                 ptrBuilding.set_floor_req(current_dir, current_floor, false);
88:                 if (loading_timer == 0)                     //посадка
89:                     loading_timer = 3;
90:                 return;
91:             }
92:
93:                                 /*если запрос "вниз" с этого этажа и если мы едем вниз или стоим,
                                                                посадить пассажиров*/
94:     if (ptrBuilding.get_floor_req(Direction.DOWN, current_floor) && current_dir != Direction.UP)
95:             {
96:                 current_dir = Direction.DOWN;                   //(если была остановка)
97:                                                    //удалить ЗЭ в данном направлении
98:                 ptrBuilding.set_floor_req(current_dir, current_floor, false);
99:                 if (loading_timer == 0)                 //посадка
100:                    loading_timer = 3;
101:                return;
102:            }
103:
104:            //проверка других назначений или ЗЭ; расстояние считать до ближайшего запроса
105:            destins_above = false;
106:            destins_below = false;
107:            for (j = current_floor + 1; j < 20; j++)                //проверка верхних этажей
108:            {               //проверить верхние этажи, если они - назначения, то установить флаг
109:                if (destination[j])
110:                    destins_above = true;
111:    if (ptrBuilding.get_floor_req(Direction.UP, j) || ptrBuilding.get_floor_req(Direction.DOWN, j))
112:                {
113:                    requests_above = true;
114:                    if (nearlest_higher_req == 0)
115:                        nearlest_higher_req = j;
116:                }
117:            }
118:            for (j = current_floor - 1; j >= 0; j--)                //проверка нижних этажей
119:            {
120:                if (destination[j])
121:                    destins_below = true;
122:    if (ptrBuilding.get_floor_req(Direction.UP, j) || ptrBuilding.get_floor_req(Direction.DOWN, j))
```

```
123:             {
124:                 requests_below = true;
125:                 if (nearlest_lower_req == 0)
126:                     nearlest_lower_req = j;
127:             }
128:         }
129:
130:                                                         //если нет запросов, остановиться
131:         if (!destins_above && !requests_above && !destins_below && !requests_below)
132:         {
133:             current_dir = Direction.STOP;
134:             return;
135:         }
136:     //если есть назначение, а мы стоим или движемся к нему, начать/продолжать движение
137:         if (destins_above && (current_dir == Direction.STOP || current_dir == Direction.UP))
138:         {
139:             current_dir = Direction.UP;
140:             return;
141:         }
142:         if (destins_below && (current_dir == Direction.STOP || current_dir == Direction.DOWN))
143:         {
144:             current_dir = Direction.DOWN;
145:             return;
146:         }
147:                                                         /*проверка, есть ли движение лифта:
                                                            (а) того же направления между нами и ближайщим ЗЭ. или
                                                            (б) встречного направления с другой стороны ЗЭ*/
148:         car_between_dn = false;
149:         car_between_up = false;
150:         car_opposite_dn = false;
161:         car_opposite_up = false;
162:         for (j = 0; j < 4; j++)                        //проверить каждый лифт
163:         {
164:             if (j != car_number)                                //если это не наш лифт
165:             {
166:                 ofloor = ptrBuilding.get_cars_floor(j);         //получить его этаж
167:                 odir = ptrBuilding.get_cars_dir(j);             //и направление
168:                                                     //если едет вверх и ЗЭ вверху
169:                 if ((odir == Direction.UP || odir == Direction.STOP) && requests_above)
170:                                         //если он при этом между нами и ЗЭ или там же, но его номер ниже
171: if ((ofloor > current_floor && ofloor <= nearlest_higher_req) ||
     (ofloor == current_floor && j < car_number))       car_between_up = true;
172:
173:                                                         //если он едет вниз и ЗЭ внизу
174:                 if ((odir == Direction.DOWN || odir == Direction.STOP) && requests_below)
175:             //если он снизу, но над ближайшим ЗЭ, или на том же этаже, но с меньшим номером
176: if ((ofloor < current_floor && ofloor >= nearlest_lower_req) ||
     (ofloor == current_floor && j < car_number))        car_between_dn = true;
177:
178:                                                         //если он имет вверх и ЗЭ снизу
179:                 if ((odir == Direction.UP || odir == Direction.STOP) && requests_below)
180:                                                         //и он ниже ЗЭ и ближе к нему, чем мы
181: if (nearlest_lower_req >= ofloor && nearlest_lower_req - ofloor < current_floor -
```

```
    nearlest_lower_req)       car_opposite_up = true;
182:
183:                                                    //если идет вниз, а ЗЭ сверху
184:                if ((odir == Direction.DOWN || odir == Direction.STOP) && requests_above)
185:                                                    //и он над ЗЭ ближе к нему, чем мы
186: if (nearlest_lower_req <= ofloor && ofloor - nearlest_lower_req < nearlest_lower_req -
    current_floor)                car_opposite_dn = true;
187:
188:            }
189:        }
190:
191:                                                    /* если идем вверх или остановились, а ЗЭ над нами
                                                        и между нами и ЗЭ нет идущих вверх лифтов
                                                        или идущих вниз над ЗЭ и ближе к нему, чем мы,
                                                        тогда ехать вверх*/
192:                    if ((current_dir == Direction.UP || current_dir == Direction.STOP) &&
                            requests_above && !car_between_up && !car_opposite_dn)
193:        {
194:            current_dir = Direction.UP;
195:            return;
196:        }
197:                                                    /* если идем вниз или остановились, и снизу есть ЗЭ
                                                        и нет лифтов, идущих вниз, между нами и ЗЭ
                                                        или под ЗЭ, идущих вверх и ближе нас к ЗЭ */
198:                if ((current_dir == Direction.DOWN || current_dir == Direction.STOP) &&
                            requests_below && !car_between_dn && !car_opposite_up)
199:        {
200:            current_dir = Direction.DOWN;
201:            return;
202:        }
203:                                                    //если ничего больше не происходит, то остановиться
204:        current_dir = Direction.STOP;
205:
206:    }                                   конец метода decide  (строки 45…206)
207:
208:  private void dests_display()    //вывести конечные этажи, выбраные кнопками внутри лифта
209:    {
210:        for (int j = 0; j < 20; j++)
211:        {
212:            Console.SetCursorPosition(7 - 2 + (car_number + 1) * 7, 20 - j);
213:            if (destination[j])
214:                Console.Write("*");                 //звездочка - обозначение этажа назначения
215:            else
216:                Console.Write(" ");                             //пробел
217:        }
218:    }
219:
220:    private void car_display()                              //вывод образа лифта
221:    {
222:        Console.SetCursorPosition(7 + (car_number + 1) * 7, 20 - old_floor);
223:        Console.Write("   ");                               //стереть со старой позиции
224:        Console.SetCursorPosition(7 - 1 + (car_number + 1) * 7, 20 - current_floor);
225:        switch (loading_timer)                                  // загрузка лифта
```

```
226:            {
227:                case 3:
228:                    Console.Write("\x01W W ");                  //лифт с открытыми дверями
229:                    break;                                      //слева - мордочка
230:                case 2:
231:                    Console.Write(" W\x01W ");                  //мордочка в дверях
232:                    get_distinations();                         //получить конечный этаж
233:                    break;
234:                case 1:
235:                    Console.Write(" WWW ");     //нарисовать с закрытыми дверями без мордочки
236:                    break;
237:                case 0:
238:                    Console.Write(" WWW ");      //двери закрыты. Мордочки нет (по умолчанию)
239:                    break;
240:            }
241:            Console.SetCursorPosition(7 + (car_number + 1) * 7, 20 - current_floor);
242:            switch (unloading_timer)                            // выгрузка лифта
243:            {
244:                case 3:
245:                    Console.Write("W\x01W ");                   //двери открыты, мордочка в дверях
246:                    break;
247:                case 2:
248:                    Console.Write("W W\x01");                   //двери открыты, мордочка справа
249:                    break;
250:                case 1:
261:                    Console.Write("WWW ");                      //двери закрыты, мордочки нет
262:                    break;
263:                case 0:
264:                    Console.Write("WWW ");          //двери закрыты, мордочки нет (по умолчанию)
265:                    break;
266:            }
267:            old_floor = current_floor;                          //запомнить старый этаж: приехали!!!
268:        }
269:
270:        public void car_tick2()                                 //метроном-2 для каждого лифта
271:        {
272:            move();                                             //двигать лифт, если нужно; с. 275…284
273:        }
274:
275:        private void move()
276:        {
277:            if (loading_timer > 0 || unloading_timer > 0)       //если посадка или высадка
278:                return;//не двигаться
279:            if (current_dir == Direction.UP)                    //если идем вверх, идти вверх
280:                current_floor++;
281:            else
282:                if (current_dir == Direction.DOWN)              //если идем вниз, идти вниз
283:                    current_floor--;
284:        }
285:
286:        public int get_floor()                                  //получение текущего этажа
287:        {
288:            return current_floor;
```

```
289:         }
290:
291:
292:         public Direction get_direction()                    //получение текущего направления
293:         {
294:             return current_dir;
295:         }
296:
297:         public void get_distinations()                      //остановка, получение этажей назначения
298:         {                                   // эквивалентно нажатию кнопок в лифте
299:             int dest_floor;                                 //этаж назначения
300:             Console.SetCursorPosition(1, 22);               //удалить вверхнюю строку
301:             Program.clear_line();
302:             Console.SetCursorPosition(1, 22);
303:     Console.Write("Лифт " + (car_number + 1) + " остановился на этаже " + (current_floor + 1) +
                        "\n Введите номера этажей назначения (или 0 для окончания ввода)");
304:             for (int j = 1; j < 20; j++)                    //получить запросы этажей
305:             {
306:                 Console.SetCursorPosition(1, 24);
307:                 Console.Write("Этаж назначения " + j + ": ");
308:                 dest_floor = int.Parse(Console.ReadLine());
309:                 Console.SetCursorPosition(1, 24);
310:                 Program.clear_line();                       //стереть старую строку
311:                 if (dest_floor == 0)                        //если больше нет запросов
312:                 {                                           //стереть нижние три строки
313:                     Console.SetCursorPosition(1, 22);
314:                     Program.clear_line();
315:                     Console.SetCursorPosition(1, 23);
316:                     Program.clear_line();
317:                     Console.SetCursorPosition(1, 24);
318:                     Program.clear_line();
319:                     return;
320:                 }
321:                 --dest_floor;                               //начинать с 0, а не 1
322:                 if (dest_floor == current_floor)            //выбрать текущий этаж
323:                 {
324:                     j--;                                    //забыть его
325:                     continue;
326:                 }
327:                 if (j == 1 && current_dir == Direction.STOP)        //записать выбор
328:                     current_dir = (dest_floor < current_floor) ? Direction.DOWN : Direction.UP;
329:                 destination[dest_floor] = true;
330:                 dests_display();                                    //вывести этажи назначения
331:             }
332:         }
333:     }
334: }                                       //окончание класса Elevator
```