Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

ОО проект:
Компьютерная модель Ипподромные состязания

Проф. Забудский Е.И.

Москва 2008

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

ОО проект:
Компьютерная модель Ипподромные состязания
(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

Компьютерная модель Ипподромные состязания

Разработана компьютерная модель «**Ипподромные состязания**». Номер лошади (H0H, H1H, H2H, …) указывается на экране дисплея. Лошадь стартует слева и скачет до финишной линии справа.

Скорость каждой лошади выбрана случайным образом, поэтому невозможно вычислить заранее, какая из них выиграет. Лошади легко, хотя и немного грубо (H0H, H1H, H2H, …), отражаются на дисплее.

При запуске программа **HORSE** запрашивает у пользователя данные о дистанции (**length**) скачек и количестве лошадей (**total**), которые принимают в них участие. Классическим участком дистанции в лошадиных скачках (по крайней мере, в *англо*язычных странах) является **1 / 8** мили (**1609 / 8 ≈ 201 м**). Обычно дистанция включает в себя **6**, **8**, **10** или **12** таких участков. Бежать могут от **1** до **7** лошадей. Программа отображает границы каждого участка скачки (**участок ≈ 201 м**) вертикальными линиями " | ", а так же линии старта и финиша линиями "] ". Каждая лошадь отображается двумя буквами "H#H" со значением номера посередине. На рис. 1 показана работа программы.

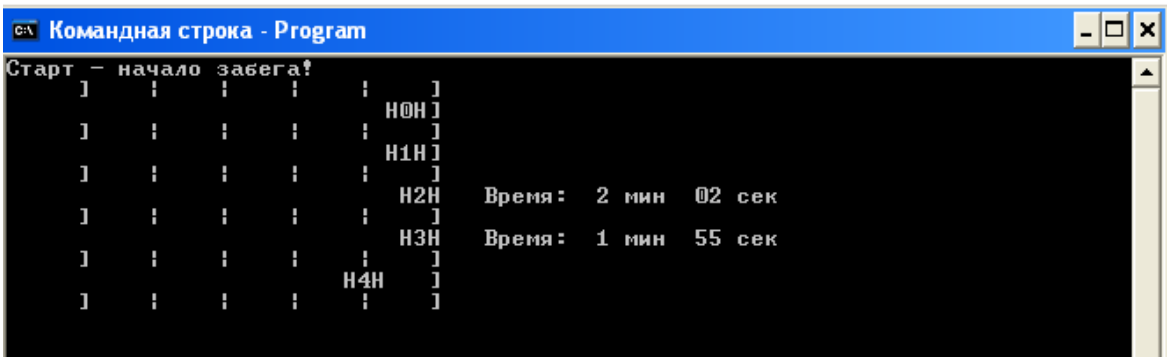

Рис. 1. Работа программы **HORSE.** Лошади H2H и H3H пришли к финишу.

1. Разработка компьютерной модели Ипподромных состязаний

Как можно применить **ООП** для программы?

Первый вопрос заключается в том, существует ли в программе группа похожих объектов, с которыми мы будем работать? Ответ положительный, это лошади. Кажется разумным представить каждую **лошадь как объект**. Создадим класс **Horse**, в котором будут содержаться данные для каждой из лошадей, такие, как: **1) ее номер** и **2) дистанция, на которой эта лошадь была самой быстрой**.

Есть еще данные, которые имеют отношение к маршруту скачек. Они включают в себя: **1) длину дистанции**, **2) время, затраченное на ее прохождение в минутах и секундах (0 мин 00 сек на старте)**, и **3) общее количество лошадей**. Поэтому имеет смысл **создать объект, который будет одиночным членом** класса **Track**. Имеются и другие объекты, которые ассоциируются с лошадиными скачками, например жокеи или конюхи, но они не будут использоваться в программе.

Существуют ли другие способы разработки программы?

Например, использование наследования для того, чтобы сделать класс **Horse** производным класса **Track**? В нашем случае это не имеет смысла, так как лошади не являются чем-то, имеющим отношение к дистанции скачек; это совсем разные вещи. **Другая возможность** — переделать данные дистанции в **статические** данные класса **Horse**. Однако **обычно лучше использовать отдельный класс для каждой предметной области**. Это выгоднее, так как позволяет упростить использования классов для других целей, например при использовании класса **Track в автомобильных гонках (другая предметная область)**.

Как же будут взаимодействовать **объекты** классов **Track** и **Horse**? (Или, в терминах **UML**, в чем состоит их связь?) Массив объектов класса **Horse** может быть членом класса **Track**, при этом **track**

сможет иметь доступ к **horse (строка 10, класс Track)**. При создании **track** будут созданы и **horse**.

На рис. 2 … рис. 4 представлены диаграммы классов **Track, Horse и Program**

## строк	Track	Класс **Track** на с. 11, 12
10…15	-elapsed_time	Текущее время забега лошади
-“-	-horse_count	Счетчик количества лошадей: текущее значение
-“-	-horses	Динамический массив лошадей
-“-	-ran	Случайное
-“-	-total_horses	Количество (беговых дорожек) лошадей, участвующих в забеге
-“-	-track_length	Длина дистации забега: она выражена количество участков длиной 201 м
37…50	+DisplayTrack()	Отображение ипподрома: дорожки и лошади, старт, финиш и участки длиной 201 м
83…86	+GetRandom()	Получить случайное значение
32…35	+GetTrackLength()	Получить длину забега
63…81	+Run()	Реализация забега
17…30	+Track()	Конструктор

Рис. 2. Диаграмма класса **Track**

## строк	Horse	Класс **Horse** на с. 12, 13
9…13	-distance_run	Текущее значение дистанции, пробегаемой лошадью
-“-	-end_run	Признак окончания забега
-“-	-finish_time	Время финиша лошади
-“-	-horse_number	Номер лошади
-“-	-ptrTrack	Указывает на Track
23…28	+EndRun {get; set}	Свойство: значение признака окончания забега
30…62	+DisplayHorse()	Перемещение лошади в новую позицию на экране и ее отображение
15…21	+Horse()	Конструктор

Рис. 3. Диаграмма класса **Horse**

## строк	Program	Класс **Program** на с. 10
9	-cpf	Кол-во ед-ц дистанции (201 м) содержащихся в 1 символе строки
10	-max_horses	Мах-е кол-во дорожек (лошадей) на ипподроме (в забеге)
33…38	+CPF{get; set}	Свойство
40…45	+MaxHorses{get; set}	Свойство
12…31	+Main()	

Рис. 4. Диаграмма класса **Program**

2. Моделирование хода времени

Программа обычно включает в себя действия, происходящие в определенный период времени. Для моделирования хода времени программа включает в себя фиксированные интервалы. В программе **HORSE** метод **Main() (класс Program)** вызывает метод **Run() (строка 25)** класса **Track (строки 63…81)**. Он делает серию вызовов метода **DisptayHorse() (строки 63…81, класс Horse)** внутри цикла **while (строки 65…80,** класс **Track)**, по одному для каждой лошади. Этот метод предназначен для

перемещения лошади в новую позицию на экране. Затем цикл **while** делает паузу на **500 миллисекунд** с помощью стандартного метода **Sleep()** **(строка 77,** класс **Track)**. Далее процесс повторяется до тех пор, пока скачки не будут окончены или пользователь не нажмет кнопку.

3. Диаграммы UML

Рассмотрим диаграмму классов **UML** для программы **HORSE.** Она показана на **рис. 5.** На этой диаграмме представлена концепция **UML,** которая называется **многообразием.**

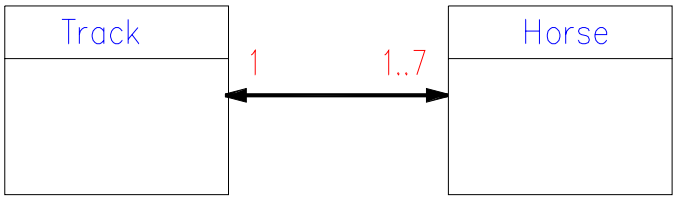

Рис. 5. UML-диаграмма классов программы HORSE

Иногда одному объекту класса **A** соответствует один объект класса **B.** В других ситуациях мы можем объединить несколько объектов класса. Это множество называется **многообразием**. Количество объектов, входящих в многообразие, обозначается на диаграмме с учетом **табл. 1.**

Таблица 1. Обозначения в многообразии UML

Символ	Значение
1	**Один**
*	**Несколько (от 0 до бесконечности)**
0..1	**Один или ни одного**
1..*	**Хотя бы один**
2..4	**2, 3 или 4**
7, 11	**7 или 11**

Если на диаграмме около класса **A** указано число **1,** а возле класса **B** символ ***,** то это будет обозначать, что **один объект** класса **A** может взаимодействовать с **произвольным количеством объектов** класса B.
В программе **HORSE** с одним объектом класса **Track** могут взаимодействовать до **7** объектов класса **Horse**. Это обозначено цифрами **1** у класса **Track** и **1..7** у класса **Horse**. Предполагаем, что в скачках может принимать участие и одна лошадь, например во время тренировок.

Между объектами классов **Track** и **Horse** существует отношение ассоциации **(“has a” – содержит)**. Ассоциация показывает, что один объект: **1)** содержит другой или **2)** связан с другим **(“has a” – содержит)** **/см. лекция 12 и 13, с. 17…27**/.

4. Диаграмма состояний в UML

В этом разделе познакомимся с новым типом диаграмм **UML**: с диаграммами состояний. **Многообразие** в **диаграммах классов UML** показывает **количество объединенных объектов**.

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.

На диаграммах классов UML отражены **взаимоотношения между классами**. В диаграмме классов **отражена организация кода** программы. 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.

С концепцией **состояния** встречаемся в нашей повседневной жизни. Радио, например, имеет два состояния: включенное и выключенное. Стиральная машина имеет такие состояния как стирка, полоскание, отжим и остановка. Для телевизора характерны состояния для каждого из его каналов (канал 7 включен и т. д.).

Между состояниями существуют **переходы**. Через 20 минут полоскания машина переходит в режим отжима. Получив сигнал от пульта управления, телевизор переключается из активного состояния канала 7 в активное состояние канала 2.

На рис. 6 показана диаграмма состояния для программы **HORSE**. На ней отражены различные состояния объекта класса **Horse**, которые он может принимать во время работы программы.

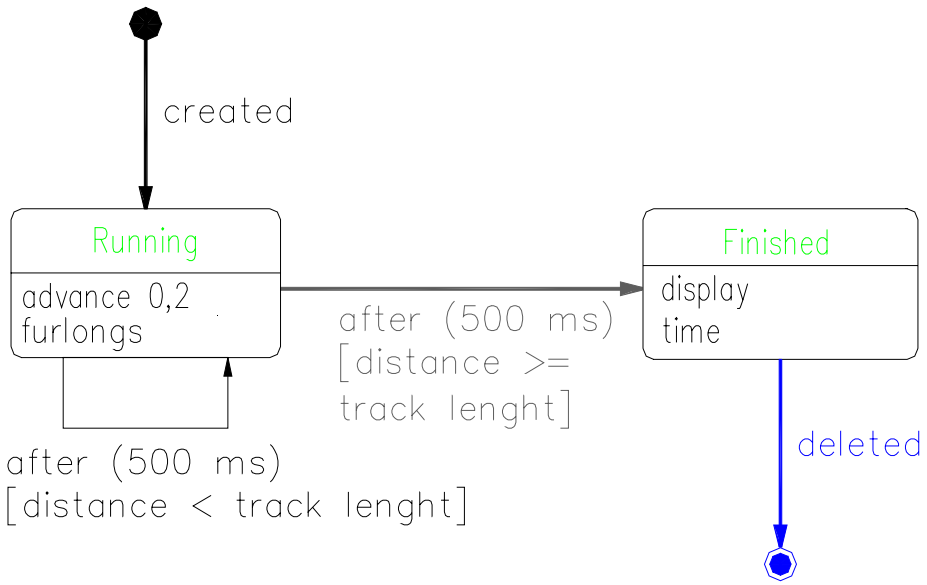

Рис. 6. Диаграмма состояний объекта класса Horse

4.1. 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.

На диаграмме присутствуют два специальных состояния: сплошным черным диском обозначено **начальное состояние**, а черным диском, помещенным в окружность, обозначено **конечное состояние**.

После создания объект класса **Horse** может пребывать только в двух состояниях: до финишной линии — состояние **Running** и после ее достижения — состояние **Finished**.

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.

4.2. 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. В нашем случае так на рис. 6 обозначены переходы **created** и **deleted**. Имя перехода не

обозначают с заглавной буквы. Имена могут быть более приближены к реальному языку, чем к терминам C#.

Событие инициирует два других перехода по истечению периода времени **500 ms**. Слово **after** (**через**) использовано как имя для этих переходов с интервалом времени в качестве параметра.

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. Оба перехода имеют защиту и имя события. Так как событие одинаковое, то условие защиты определяет, какой из переходов будет выполнен.

Заметим, что один из переходов является переходом сам в себя, он возвращает нас в то же состояние.

4.3. От состояния к состоянию

Каждый раз, попадая в состояние **Running**, объект класса **Horse** выполняет действие, заключающееся в увеличении пройденного расстояния на **0.2** участка дистанции. Пока мы не достигли финиша, будет выполняться условие защиты [**distance < track length**], и мы будем возвращаться в состояние **Running**. Когда лошадь достигнет финиша, выполнится условие защиты [**distance >= track length**], и объект перейдет в состояние **Finished**, где будет выведено время скачки. Затем объект можно удалить (**очистить память**).

Вопросы и ответы

1. Операция **new (строка 24, класс Program)**:

а) автоматически вызывает контструктор Track() класса Track;
б) создает новый объект **track** класса Track;
в) выполняет инициализацию переменных;
г) обеспечивает выделение памяти для объекта **track**.

2. **Связный список** — это:

а) структура, где каждый элемент представляет собой указатель на следующий элемент;
б) массив указателей, указывающих на элементы списка;
в) структура, в которой каждый элемент состоит: 1) из данных или 2) указателя на данные;
г) структура, в которой элементы хранятся в массиве.

3. Если мы хотим отсортировать множество больших объектов или структур, то будет более эффективным:

а) поместить их в массив и сортировать как его элементы;
б) создать массив указателей на них и отсортировать его;
в) поместить эти объекты в связный список и отсортировать его;
г) поместить ссылки на эти объекты в массив и отсортировать его.

4. Изобразите многообразие объединений, которые имеют до 10 объектов с одной стороны и больше двух — с другой стороны. **Ответ: 0..9 с одного конца; 3..* — с другого.**

5. Состояния в диаграмме состояний соответствуют:

а) сообщениям между объектами;
б) условиям, по которым объекты находят себя;
в) объектам программы;
г) изменениям ситуации, в которой используются объекты.

6. Истинно ли следующее утверждение: переходы между состояниями существуют во время исполнения программы? **Ответ: Ложно**

7. Защита в диаграмме состояний — это:

а) ограничивающее условие на переход;

б) имя определенного перехода;

в) имя определенного состояния;

г) ограничение на создание определенных состояний.

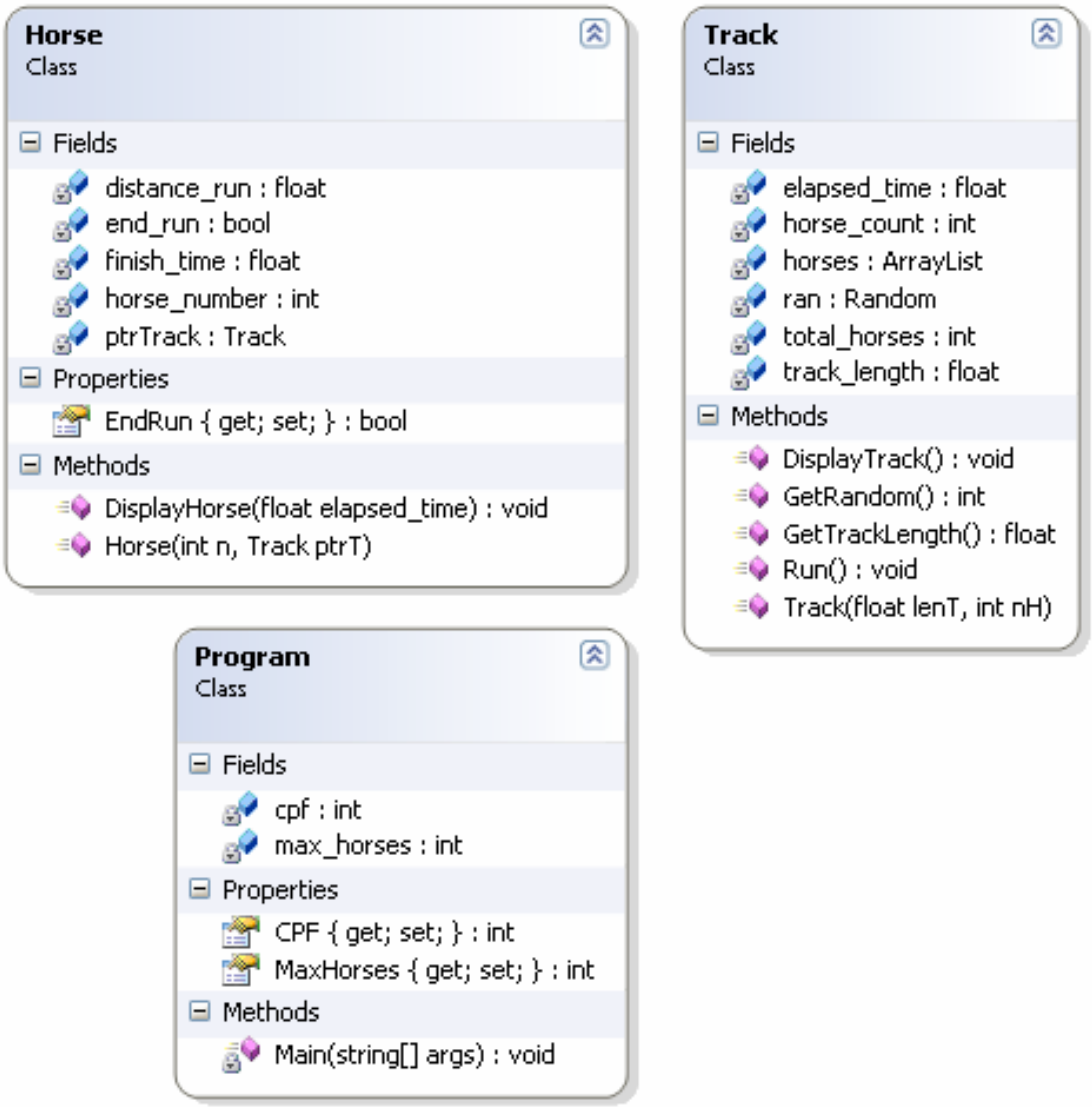

Рис. 7. Диаграмма классов программы Horse

Задание студентам

1. Разобраться в коде программы Horse.
2. Реализовать WINDOWS-интерфейс программы Horse. **Дерзайте!!**
3. Вывести статистические данные: количество лошадей, участвующих в забеге;
   время финиша каждой лошади.

```
1:  using System;
2:  using System.Collections.Generic;
3:  using System.Text;
4:
5:  namespace Horse
6:  {
7:      class Program                                   // класс Program
8:      {
9:      const int cpf = 5;  //сколько единиц  дистанции содержится в одном  ASCII-символе строки
10:         const int max_horses = 7;  //количество дорожек на ипподроме
11:
12:         static void Main(string[ ] args)
13:         {
14:             float length;         // длина дистанции
15:             int total;            //количество лошадей, участвующих в забеге
16:             string input_c;     // новый забег
17:             do
18:             {
19:                 Console.Clear();                  //вводим начальные данные
20:          Console.Write("Введите длину дистанции X (от 1 до 12): "); // X·201 м
21:                 length = int.Parse(Console.ReadLine());
22:                 Console.Write("Введите количество лошадей (от 1 до 7): ");
23:                 total = int.Parse(Console.ReadLine());
24:                 Track track = new Track(length, total);              // строки 17…30
25:                 track.Run();           //Старт - начало забега!        Строки 63…81
26:                                        //Предлагаем еще забег
27:         Console.Write("\n\nФиниш - забег закончен! Начать новый? (Y/N): ");
28:                 input_c = Console.ReadLine();
29:             }
30:             while (input_c == "Y" || input_c == "y");
31:         }
32:
33:         public static int CPF   // свойство
34:         {
35:             get
36:             { return cpf; }
37:             set { }
38:         }
39:
40:         public static int MaxHorses   // свойство
41:         {
42:             get
43:             { return max_horses; }
44:             set { }
45:         }
46:     }                               // конец класса Program
47: }
```

```
1:  using System;
2:  using System.Collections.Generic;
3:  using System.Collections;
4:  using System.Text;
5:
6:  namespace Horse
7:  {
8:      public class Track                                  // класс Track
9:      {
10:         private ArrayList horses;                               // динамический массив лошадей
11:         private int total_horses;
12:         private int horse_count;
13:         private float track_length;
14:         private float elapsed_time;
15:         private Random ran;
16:
17:         public Track(float lenT, int nH)                //конструктор для трека
18:         {
19:             ran = new Random();                         // реализуем генератор случайных чисел
20:             track_length = lenT;
21:             total_horses = nH;
22:             horse_count = 0;
23:             elapsed_time = 0f;
24:                                                         //проверяем: уместятся все лошади на дорожках?
25:     total_horses = (total_horses > Program.MaxHorses) ? Program.MaxHorses : total_horses;
26:             horses = new ArrayList();
27:             for (int j = 0; j < total_horses; j++)
28:                 horses.Add(new Horse(horse_count++, this));   // строки 15…21 класса Horse
29:             DisplayTrack();                 // строки 37…61
30:         }
31:
32:         public float GetTrackLength()
33:         {
34:             return track_length;
35:         }
36:
37:         public void DisplayTrack()
38:         {                                                   //отображаем полоски трека
39:             Console.Clear();
40:             for (int f = 0; f <= track_length; f++)
41:             {
42:                 for (int r = 1; r <= total_horses * 2 + 1; r++)
43:                 {
44:                     Console.SetCursorPosition(f * Program.CPF + 5, r);
45:                     if (f == 0 || f == track_length)
46:                         Console.Write("]");                     // отображение старта и финиша
47:                     else
48:                         Console.Write("|");     // отображение границ между участками (участок ≈201 м)
49:                 }
50:             }
```

```
61:         }
62:
63:         public void Run()    // реализация забега на заданную дистанцию
64:         {                                   //повторяющийся цикл, имитирующий промежуток времени
65:             while (true)
66:             {
67:                 elapsed_time += 1.75f;
68:                 int ended_horses = 0;
69:                                                                         //показываем каждую лошадь
70:                 for (int j = 0; j < total_horses; j++)
71:                 {
72:                     Horse h = (Horse) horses[j];
73:                     h.DisplayHorse(elapsed_time);              // строки 30…62 класса Horse
74:       if (h.EndRun)  ended_horses++;    //если лошадь закончила забег, то увеличиваем счетчик
75:                  }
76:
77:                 System.Threading.Thread.Sleep(500);                 //500 ms - задержка между шагами
78:           if (ended_horses == total_horses)  return;  //если забег закончен, то выходим из метода
79:
80:              }
81:         }
82:
83:         public int GetRandom()
84:         { // только при возврате значения «1»– лошадь перемещается на экране: строки 32 и 37 кл. Horse
85:             return ran.Next(1, 4);         //генерация случайного int-числа  в диапазоне 1…4
86:         }
87:     }                                       // конец класса Track
88: }
```

```
1:  using System;
2:  using System.Collections.Generic;
3:  using System.Text;
4:
5:  namespace Horse
6:  {
7:      public class Horse                                  // класс Horse
8:      {
9:          private Track ptrTrack;
10:         private int horse_number;
11:         private float finish_time;
12:         private float distance_run;
13:         private bool end_run;
14:
15:         public Horse(int n, Track ptrT)                             //конструктор
16:         {
17:             this.horse_number = n;
18:             this.ptrTrack = ptrT;
19:             distance_run = 0f;
20:             end_run = false;
21:         }
22:
```

```
23:         public bool EndRun
24:         {
25:             get
26:             { return end_run; }
27:             set {    }
28:         }
29:       // метод DisplayHorse предназначен для перемещения лошади в новую позицию на экране
30:         public void DisplayHorse(float elapsed_time)
31:         {                   /*метод вычисляет: будет ли лошадь двигаться? И выводит ее на экран.
                                                                  Вывод лошади на экран*/
32:             Console.SetCursorPosition(1 + (int)(distance_run * Program.CPF), 2 + horse_number * 2);
33:             Console.Write(" H" + horse_number + "H");
34:                                         /*если лошадь еще не закончила забег, и выпал 1 шанс из 3-х,
                                              то увеличиваем дистанцию, которую пробежала лошадь */
35:             if (distance_run < ptrTrack.GetTrackLength() + 1f / Program.CPF) // если =, то дистанция пройдена
36:             {
37:                 if (ptrTrack.GetRandom() == 1)  distance_run += 0.2f;       // перемещение лошади
38:
39:                 finish_time = elapsed_time;
40:             }
41:                                         //если лошадь закончила забег, то печатаем время окончания
42:             else
43:             {
44:                 int mins = (int)(finish_time / 60);
45:                 int secs = (int)(finish_time - mins * 60);
46:                 string additional_zero = "";
47:                 if (secs < 10)   additional_zero = "0";
48:
49:                 Console.Write("   Время:  " + mins + " мин  " + additional_zero + secs + " сек ");
50:                 end_run = true;   // лошадь прошла всю дистанцию забега
61:             }
62:         }             // конец метода DisplayHorse
63:     }                                                          // конец класса Horse
64: }
```