**Делегаты**

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты - это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

**Определение делегатов**

Для объявления делегата используется ключевое слово **delegate**, после которого идет возвращаемый тип, название и параметры. Например:

delegate void Message();

Делегат Message в качестве возвращаемого типа имеет тип void (то есть ничего не возвращает) и не принимает никаких параметров. Это значит, что этот делегат может указывать на любой метод, который не принимает никаких параметров и ничего не возвращает.

Рассмотрим примение этого делегата:

class Program

{

delegate void Message(); // 1. Объявляем делегат

static void Main(string[] args)

{

Message mes; // 2. Создаем переменную делегата

if (DateTime.Now.Hour < 12)

{

mes = GoodMorning; // 3. Присваиваем этой переменной адрес метода

}

else

{

mes = GoodEvening;

}

mes(); // 4. Вызываем метод

Console.ReadKey();

}

private static void GoodMorning()

{

Console.WriteLine("Good Morning");

}

private static void GoodEvening()

{

Console.WriteLine("Good Evening");

}

}

В данном случае делегат определяется внутри класса, но также можно определить делегат вне класса внутри пространства имен.

С помощью свойства DateTime.Now.Hour получаем текущий час. И в зависимости от времени в делегат передается адрес определенного метода. Обратите внимание, что методы эти имеют то же возвращаемое значение и тот же набор параметров (в данном случае отсутствие параметров), что и делегат.

Затем через делегат вызываем метод, на который ссылается данный делегат:

Вызов делегата производится подобно вызову метода.

Посмотрим на примере другого делегата:

class Program

{

delegate int Operation(int x, int y);

static void Main(string[] args)

{

// присваивание адреса метода через контруктор

Operation del = Add; // делегат указывает на метод Add

int result = del(4,5); // фактически Add(4, 5)

Console.WriteLine(result);

del = Multiply; // теперь делегат указывает на метод Multiply

result = del(4, 5); // фактически Multiply(4, 5)

Console.WriteLine(result);

Console.Read();

}

private static int Add(int x, int y)

{

return x+y;

}

private static int Multiply (int x, int y)

{

return x \* y;

}

}

В данном случае делегат Operation возвращает значение типа int и имеет два параметра типа int. Поэтому этому делегату соответствует любой метод, который возвращает значение типа int и принимает два параметра типа int. В данном случае это методы Add и Multiply. То есть мы можем присвоить переменной делегата любой из этих методов и вызывать.

Поскольку делегат принимает два параметра типа int, то при его вызове необходимо передать значения для этих параметров: del(4,5).

Делегаты необязательно могут указывать только на методы, которые определены в том же классе, где определена переменная делегата. Это могут быть также методы из других классов и структур.

class Math

{

public int Sum(int x, int y) { return x + y; }

}

class Program

{

delegate int Operation(int x, int y);

static void Main(string[] args)

{

Math math = new Math();

Operation del = math.Sum;

int result = del(4, 5); // math.Sum(4, 5)

Console.WriteLine(result); // 9

Console.Read();

}

}

**Присвоение ссылки на метод**

Выше переменной делегата напрямую присваивался метод. Есть еще один способ - создание объекта делегата с помощью конструктора, в который передается нужный метод:

class Program

{

delegate int Operation(int x, int y);

static void Main(string[] args)

{

Operation del = Add;

Operation del2 = new Operation(Add);

Console.Read();

}

private static int Add(int x, int y) { return x + y; }

}

Оба способа равноценны.

**Соответствие методов делегату**

Как было написано выше, методы соответствуют делегату, если они имеют один и тот же возвращаемый тип и один и тот же набор параметров. Но надо учитывать, что во внимание также принимаются модификаторы **ref** и **out**. Например, пусть у нас есть делегат:

delegate void SomeDel(int a, double b);

Этому делегату соответствует, например, следующий метод:

void SomeMethod1(int g, double n) { }

А следующие методы НЕ соответствуют:

int SomeMethod2(int g, double n) { }

void SomeMethod3(double n, int g) { }

void SomeMethod4(ref int g, double n) { }

void SomeMethod5(out int g, double n) { g = 6; }

Здесь метод SomeMethod2 имеет другой возвращаемый тип, отличный от типа делегата. SomeMethod3 имеет другой набор параметров. Параметры SomeMethod4 и SomeMethod5 также отличаются от параметров делегата, поскольку имеют модификаторы ref и out.

**Добавление методов в делегат**

В примерах выше переменная делегата указывала на один метод. В реальности же делегат может указывать на множество методов, которые имеют ту же сигнатуру и возвращаемые тип. Все методы в делегате попадают в специальный список - список вызова или invocation list. И при вызове делегата все методы из этого списка последовательно вызываются. И мы можем добавлять в этот спиcок не один, а несколько методов:

class Program

{

delegate void Message();

static void Main(string[] args)

{

Message mes1 = Hello;

mes1 += HowAreYou; // теперь mes1 указывает на два метода

mes1(); // вызываются оба метода - Hello и HowAreYou

Console.Read();

}

private static void Hello()

{

Console.WriteLine("Hello");

}

private static void HowAreYou()

{

Console.WriteLine("How are you?");

}

}

В данном случае в список вызова делегата mes1 добавляются два метода - Hello и HowAreYou. И при вызове mes1 вызываются сразу оба этих метода.

Для добавления делегатов применяется операция **+=**. Однако стоит отметить, что в реальности будет происходить создание нового объекта делегата, который получит методы старой копии делегата и новый метод, и новый созданный объект делеагата будет присвоен переменной mes1.

При добавлении делегатов следует учитывать, что мы можем добавить ссылку на один и тот же метод несколько раз, и в списке вызова делегата тогда будет несколько ссылок на один и то же метод. Соответственно при вызове делегата добавленный метод будет вызываться столько раз, сколько он был добавлен:

Message mes1 = Hello;

mes1 += HowAreYou;

mes1 += Hello;

mes1 += Hello;

mes1();

Hello

How are you?

Hello

Hello

**Объединение делегатов**

Делегаты можно объединять в другие делегаты. Например:

class Program

{

delegate void Message();

static void Main(string[] args)

{

Message mes1 = Hello;

Message mes2 = HowAreYou;

Message mes3 = mes1 + mes2; // объединяем делегаты

mes3(); // вызываются все методы из mes1 и mes2

Console.Read();

}

private static void Hello()

{

Console.WriteLine("Hello");

}

private static void HowAreYou()

{

Console.WriteLine("How are you?");

}

}

В данном случае объект mes3 представляет объединение делегатов mes1 и mes2. Объединение делегатов значит, что в список вызова делегата mes3 попадут все методы из делегатов mes1 и mes2. И при вызове делегата mes3 все эти методы одновременно будут вызваны.

**Вызов делегата**

В примерах выше делегат вызывался как обычный метод. Если делегат принимал параметры, то при ее вызове для параметров передавались необходимые значения:

class Program

{

delegate int Operation(int x, int y);

delegate void Message();

static void Main(string[] args)

{

Message mes = Hello;

mes();

Operation op = Add;

op(3, 4);

Console.Read();

}

private static void Hello() { Console.WriteLine("Hello"); }

private static int Add(int x, int y) { return x + y; }

}

Другой способ вызова делегата представляет метод **Invoke()**:

class Program

{

delegate int Operation(int x, int y);

delegate void Message();

static void Main(string[] args)

{

Message mes = Hello;

mes.Invoke();

Operation op = Add;

op.Invoke(3, 4);

Console.Read();

}

private static void Hello() { Console.WriteLine("Hello"); }

private static int Add(int x, int y) { return x + y; }

}

Если делегат принимает параметры, то в метод Invoke передаются значения для этих параметров.

Следует учитывать, что если делегат пуст, то есть в его списке вызова нет ссылок ни на один из методов (то есть делегат равен Null), то при вызове такого делегата мы получим исключение, как, например, в следующем случае:

Message mes = null;

//mes(); // ! Ошибка: делегат равен null

Operation op = Add;

op -= Add; // делегат op пуст

op(3, 4); // !Ошибка: делегат равен null

Поэтому при вызове делегата всегда лучше проверять, не равен ли он null. Либо можно использовать метод Invoke и оператор условного null:

Message mes = null;

mes?.Invoke(); // ошибки нет, делегат просто не вызывается

Operation op = Add;

op -= Add; // делегат op пуст

op?.Invoke(3, 4); // ошибки нет, делегат просто не вызывается

Если делегат возвращает некоторое значение, то возвращается значение последнего метода из списка вызова (если в списке вызова несколько методов). Например:

class Program

{

delegate int Operation(int x, int y);

static void Main(string[] args)

{

Operation op = Subtract;

op += Multiply;

op += Add;

Console.WriteLine(op(7, 2)); // Add(7,2) = 9

Console.Read();

}

private static int Add(int x, int y) { return x + y; }

private static int Subtract(int x, int y) { return x - y; }

private static int Multiply(int x, int y) { return x \* y; }

}

**Делегаты как параметры методов**

Также делегаты могут быть параметрами методов:

|  |
| --- |
| class Program  {  delegate void GetMessage();    static void Main(string[] args)  {  if (DateTime.Now.Hour < 12)  {  Show\_Message(GoodMorning);  }  else  {  Show\_Message(GoodEvening);  }  Console.ReadLine();  }  private static void Show\_Message(GetMessage \_del)  {  \_del?.Invoke();  }  private static void GoodMorning()  {  Console.WriteLine("Good Morning");  }  private static void GoodEvening()  {  Console.WriteLine("Good Evening");  }  } |

**Обобщенные делегаты**

Делегаты могут быть обобщенными, например:

delegate T Operation<T, K>(K val);

class Program

{

static void Main(string[] args)

{

Operation<decimal, int> op = Square;

Console.WriteLine(op(5));

Console.Read();

}

static decimal Square(int n)

{

return n \* n;

}

}

Наиболее сильная сторона делегатов состоит в том, что они позволяют делегировать выполнение некоторому коду извне. И на момент написания программы мы можем не знать, что за код будет выполняться. Мы просто вызываем делегат. А какой метод будет непосредственно выполняться при вызове делегата, будет решаться потом. Например, наши классы будут распространяться в виде отдельной библиотеки классов, которая будет подключаться в проект другого разработчика. И этот разработчик захочет определить какую-то свою логику обработки, но изменить исходный код нашей библиотеки классов он не может. И делегаты как раз предоставляют возможность вызвать некое действие, которое задается извне и которое на момент написания кода может быть неизвестно.

Рассмотрим подробный пример. Пусть у нас есть класс, описывающий счет в банке:

class Account

{

int \_sum; // Переменная для хранения суммы

public Account(int sum)

{

\_sum = sum;

}

public int CurrentSum

{

get { return \_sum; }

}

public void Put(int sum)

{

\_sum += sum;

}

public void Withdraw(int sum)

{

if (sum <= \_sum)

{

\_sum -= sum;

}

}

}

Допустим, в случае вывода денег с помощью метода Withdraw нам надо как-то уведомлять об этом самого клиента и, может быть, другие объекты. Для этого создадим делегат **AccountStateHandler**. Чтобы использовать делегат, нам надо создать переменную этого делегата, а затем присвоить ему метод, который будет вызываться делегатом.

Итак, добавим в класс Account следующие строки:

class Account

{

// Объявляем делегат

public delegate void AccountStateHandler(string message);

// Создаем переменную делегата

AccountStateHandler \_del;

// Регистрируем делегат

public void RegisterHandler(AccountStateHandler del)

{

\_del = del;

}

// Далее остальные строки класса Account

Здесь фактически проделываются те же шаги, что были выше, и есть практически все кроме вызова делегата. В данном случае у нас делегат принимает параметр типа string. Теперь изменим метод Withdraw следующим образом:

public void Withdraw(int sum)

{

if (sum <= \_sum)

{

\_sum -= sum;

if (\_del != null)

\_del($"Сумма {sum} снята со счета");

}

else

{

if (\_del != null)

\_del("Недостаточно денег на счете");

}

}

Теперь при снятии денег через метод Withdraw мы сначала проверяем, имеет ли делегат ссылку на какой-либо метод (иначе он имеет значение null). И если метод установлен, то вызываем его, передавая соответствующее сообщение в качестве параметра.

Теперь протестируем класс в основной программе:

class Program

{

static void Main(string[] args)

{

// создаем банковский счет

Account account = new Account(200);

// Добавляем в делегат ссылку на метод Show\_Message

// а сам делегат передается в качестве параметра метода RegisterHandler

account.RegisterHandler(new Account.AccountStateHandler(Show\_Message));

// Два раза подряд пытаемся снять деньги

account.Withdraw(100);

account.Withdraw(150);

Console.ReadLine();

}

private static void Show\_Message(String message)

{

Console.WriteLine(message);

}

}

Запустив программу, мы получим два разных сообщения:

Сумма 100 снята со счета

Недостаточно денег на счете

Таким образом, мы создали механизм обратного вызова для класса Account, который срабатывает в случае снятия денег. Поскольку делегат объявлен внутри класса Account, то чтобы к нему получить доступ, используется выражение Account.AccountStateHandler.

Опять же может возникнуть вопрос: почему бы в коде метода Withdraw() не выводить сообщение о снятии денег? Зачем нужно задействовать какой-то делегат?

Дело в том, что не всегда у нас есть доступ к коду классов. Например, часть классов может создаваться и компилироваться одним человеком, который не будет знать, как эти классы будут использоваться. А использовать эти классы будет другой разработчик.

Так, здесь мы выводим сообщение на консоль. Однако для класса Account не важно, как это сообщение выводится. Классу Account даже не известно, что вообще будет делаться в результате списания денег. Он просто посылает уведомление об этом через делегат.

В результате, если мы создаем консольное приложение, мы можем через делегат выводить сообщение на консоль. Если мы создаем графическое приложение Windows Forms или WPF, то можно выводить сообщение в виде графического окна. А можно не просто выводить сообщение. А, например, записать при списании информацию об этом действии в файл или отправить уведомление на электронную почту. В общем любыми способами обработать вызов делегата. И способ обработки не будет зависеть от класса Account.

Хотя в примере наш делегат принимал адрес на один метод, в действительности он может указывать сразу на несколько методов. Кроме того, при необходимости мы можем удалить ссылки на адреса определенных методов, чтобы они не вызывались при вызове делегата. Итак, изменим в классе Account метод RegisterHandler и добавим новый метод UnregisterHandler, который будет удалять методы из списка методов делегата:

// Регистрируем делегат

public void RegisterHandler(AccountStateHandler del)

{

    \_del += del; // добавляем делегат

}

// Отмена регистрации делегата

public void UnregisterHandler(AccountStateHandler del)

{

    \_del -= del; // удаляем делегат

}

class Program

{

static void Main(string[] args)

{

Account account = new Account(200);

Account.AccountStateHandler colorDelegate = new Account.AccountStateHandler(Color\_Message);

// Добавляем в делегат ссылку на методы

account.RegisterHandler(new Account.AccountStateHandler(Show\_Message));

account.RegisterHandler(colorDelegate);

// Два раза подряд пытаемся снять деньги

account.Withdraw(100);

account.Withdraw(150);

// Удаляем делегат

account.UnregisterHandler(colorDelegate);

account.Withdraw(50);

Console.ReadLine();

}

private static void Show\_Message(String message)

{

Console.WriteLine(message);

}

private static void Color\_Message(string message)

{

// Устанавливаем красный цвет символов

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(message);

// Сбрасываем настройки цвета

Console.ResetColor();

}

}

В целях тестирования мы создали еще один метод - Color\_Message, который выводит то же самое сообщение только красным цветом. Для первого делегата создается отдельная переменная. Но большой разницы между передачей обоих в метод account.RegisterHandler нет: просто в одном случае мы сразу передаем объект, создаваемый конструктором account.RegisterHandler(new Account.AccountStateHandler(Show\_Message));

Во втором случае создаем переменную и ее уже передаем в метод account.RegisterHandler(colorDelegate);.

В строке account.UnregisterHandler(colorDelegate); этот метод удаляется из списка вызовов делегата, поэтому этот метод больше не будет срабатывать.

Задания

1. Разберитесь и протестируйте вышеописанный пример!!!

2 Сделайте делегат, рассчитывающий определенный интеграл

delegate double integr (double a, double b, double h)

c помощью делегата рассчитайте интеграл методом прямоугольников и трапеций

вместо формул сразу код

//прямоугольники

for(x=a;x<b;x=x+h)

s=s+f(x)\*h;

//трапеции

for(x=a;x<b;x=x+h)

s=s+(f(x)+f(x+h))\*h;

вызовите оба в качестве пробной функции используйте x^2;

3. создайте класс с именем integr поместите оба метода (прямоугольнике и трапеций в этот класс)

В основной программе объявите функцию ff(x)

delegate double ff(double x); (выше всех классов)

посчитайте определенный интеграл по принципу s=i.trap(a,b,h,f) где где f подынтегральная функция x^2 заданная в основном классе . i-имеет тип integr

подсказка double trap(double a, double b, double h, ff f)