Модуль 3, практическое занятие 2

Делегаты

Обратный вызов

Задача 1

- **В библиотеке классов описать:**
  - делегат **ConvertRule**, представляющий методы, возвращающие строку, с одним параметром – строкой
  - Класс **Converter** с нестатическим методом **public string Convert(string str, ConvertRule cr)**. Метод преобразует строку **str** по правилу **cr.**
- **В проекте консольного приложения описать два метода преобразования строк**:
  - **public static string RemoveDigits(string str)** – возвращает строку, полученную из **str** удалением цифр
  - **public static string RemoveSpaces(string str)** – возвращает строку, полученную из **str** удалением пробелов
  - В методе **Main()** описать массив тестовых строк, связать методы с объектом делегатом типа **ConvertRule** и протестировать работу каждого метода.
  - Связать один многоадресный делегат с обоими методами и протестировать вызовы на том же массиве строк

Задача 2

- Используем делегат и обратный вызов для работы с состоянием объектов класса **Car**, представляющего машину. С помощью делегата будем оповещать вызывающий код об изменении состояния машины при разгоне и поломке при достижении максимальной скорости.
- В вызывающем коде определим метод, который будет запускаться при оповещении.

Задача основана на примере из книги *A. Troelsen, Pro C# 5.0 and the .NET 4.5 FrameWork, p. 359*

Задача 2. Исходная версия класса Car

```
public class Car {
        // Информация о внутреннем состоянии
        public int CurrentSpeed { get; set; }
        public int MaxSpeed { get; set; }
        public string PetName { get; set; }
        // Машина работоспособна?
        private bool carIsDead;
        // Конструкторы
        public Car() { MaxSpeed = 100; }
        public Car(string name, int maxSp, int currSp) {
            CurrentSpeed = currSp;
            MaxSpeed = maxSp;
            PetName = name;
        }
    }
```

Задача 2

- **Выполним следующие действия:**
  1. Определим тип-делегат, который будет использоваться для отправки оповещений в вызывающий код
     ```
     public delegate void CarEngineHandler(string msgForCaller);
     ```
  2. Добавим в класс Car закрытое поле
     ```
     private CarEngineHandler listOfHandlers
     ```
  3. Описать вспомогательную функцию, в классе Car вспомогательную функцию, позволяющую передавать метод, который должен запускаться в вызывающем коде
     ```
     public void RegisterWithCarEngine(CarEngineHandler methodToCall) {
                     listOfHandlers = methodToCall;
     }
     ```

Задача 2

- Опишем метод, разгоняющий машину и оповещающий вызывающий код о её состоянии:
  4. Реализуйте метод **`Accelerate()`**, в котором происходят вызовы методов из вызывающего кода через делегат **`listOfHandlers`**

Задача 2

```
public void Accelerate(int delta) {
// Если машина сломана, отправляем оповещение.
if (carIsDead) {
     if (listOfHandlers != null)
          listOfHandlers("К сожалению, машина сломана :( ...");
            }
     else {
         CurrentSpeed += delta;
                // Машина почти сломана?
                if (10 == (MaxSpeed - CurrentSpeed)
                && listOfHandlers != null) {
                    listOfHandlers("Предупреждение! Будь осторожнее");

                }
                if (CurrentSpeed >= MaxSpeed)
                    carIsDead = true;
                else
                    Console.WriteLine("Скорость = {0}", CurrentSpeed);
            }
        }
```

Задача 2. Тестовое приложение

```
static void Main() {
    Console.WriteLine("***** Использование делегатов для управления событиями
*****\n");

    Car c1 = new Car("SlugBug", 100, 10);

    // Передаём в машину метод, который будет вызван при отправке оповещения.
    c1.RegisterWithCarEngine(new Car.CarEngineHandler(OnCarEngineEvent));
    // Разгоняем машину
    Console.WriteLine("***** Увеличиваем скорость *****");
    for (int i = 0; i < 6; i++)
          c1.Accelerate(20);
          Console.ReadLine();
    }
// Это метод-обработчик оповещений от машины.
public static void OnCarEngineEvent(string msg) {
    Console.WriteLine("\n***** Сообщение от объекта типа Car *****");
    Console.WriteLine("=> {0}", msg);
    Console.WriteLine("*************************************\n");
}
```

Задача 3

Создадим библиотеку классов с именем Numerical.

В библиотеке опишем метод поиска вещественного корня функции одного аргумента на заданном интервале.

Используем в качестве прототипа для решения нашей задачи алгоритм под номером 4б «Нахождение корней непрерывной функции методом деления интервала пополам» из книги «Библиотека алгоритмов 1б-50б (Справочное пособие.)» М., «Сов. радио», 1975. -176 с.

Метод бисекции (https://ru.wikipedia.org/wiki/Метод_бисекции)

Задача 3. Библиотека классов

```
public delegate double function (double x); //Объявление делегата-типа
```

```
public class NumMeth {
   // Метод поиска корня функции делением интервала пополам:
   static public double Bisec (double a, double b, // Границы интервала
      double epsX, double epsY,    // точность по абсциссе и ординате
      function f)    {  // f - исследуемая функция
      double x, y, z; // локальные переменные
      x = a; y = f (x);
      if (Math.Abs (y) <= epsY) return x;
      x = b; z = f (x);
      if (Math.Abs (z) <= epsY) return x;
      if (y * z > 0)
         throw new Exception ("Интервал не локализует корень функции!");
      do {
          x = a / 2 + b / 2; y = f (x);
          if (Math.Abs (y) <= epsY) return x;
          if (y * z > 0) b = x; else a = x;
      } while (Math.Abs (b - a) >= epsX);
      return x;
  }   // Bisec()
}
```

Задача 3

Создадим консольное приложения для тестирования библиотеки классов Numerical.
Найдем корни математических функций, используя библиотечный метод Bisec(). В качестве аргументов, заменяющих параметр-делегат, используем:

- **библиотечную функцию (метод из стандартной библиотеки),**
- **статический метод, явно определенный в программе,**
- **анонимный метод,**
- **лямбда-выражение.**

Задача 3 (продолжение)

Дополним библиотеку Numerical численным методом для поиска минимума одномерной вещественной функции.

Выберем алгоритм на основе метода золотого сечения, описанный в работе
NUMERICAL METHODS for Mathematics, Science and Engineering, 2nd Ed, 1992 Prentice Hall, Englewood Cliffs, New Jersey, 07632, U.S.A.
Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-624990-6
Prentice Hall, International Editions: ISBN 0-13-625047-5
Существует реализация алгоритма на языке Си:
NUMERICAL METHODS: C Programs, (c) John H. Mathews 1995.

Задача 3. Библиотека классов (продолжение)

```
// Делегат-тип для представления критерия оптимизации:
public delegate double Functional_1 (double x);
```

```
public static double Optimum_1 (Functional_1 fun,
  // fun - минимизируемая функция (функционал)
    double A, double B,            //границы интервала
    double Delta, double Epsilon)    { // точности по абсциссе и ординате
       double Rone = (Math.Sqrt (5.0) - 1.0) / 2.0; // Determine constants
       double Rtwo = Rone * Rone;       /* for golden search */
       double YA, YB;                   /* function values at A and B */
       double C, D;                     /* interior points */
       double H;                        /* width of interval */
       double P, YC, YD, YP;            /* minimum and function values */
       YA = fun (A);
       YB = fun (B);
       H = B - A;
       C = A + Rtwo * H;
       YC = fun (C);
       D = A + Rone * H;
       YD = fun (D);
```

Задача 3. Библиотека классов (продолжение)

```
while (Math.Abs (YA - YB) > Epsilon || H > Delta) {
                if (YC < YD) {
                    B = D;
                    YB = YD;
                    D = C;
                    YD = YC;
                    H = B - A;
                    C = A + Rtwo * H;
                    YC = fun (C);
                }
                else {
                    A = C; YA = YC;
                    C = D;
                    YC = YD;
                    H = B - A;
                    D = A + Rone * H;
                    YD = fun (D);
                }
            }
            P = A;
            YP = YA;
            if (YB < YA) { P = B; YP = YB; }
            return P;
        }//Optimum_1()
```

Задача 3. Тестовое приложение

Для тестирования метода Optimum_1 создадим консольное приложение и запрограммируем поиск минимума следующих функций (используйте лямбда выражения):

- ***cos(x)* на интервале A=3, B=6;**
- ***x*(x*x-2)-5* на интервале A=0, B=1;**
- ***-Sin(x)-Sin(3*x)/3* на интервале A = 0; B = 1;**

Задача 4

Определить класс Series, поле которого – ссылка на целочисленный массив. Метод Order класса Series выполняет сортировку массива. Для сравнения элементов используется предикат, представляемый делегатом.

В тестирующем классе определить несколько методов, играющих роли предикатов, и выполнить с их помощью упорядочивание массива в объекте класса series.

Задача 5

Объявить массив из пяти вещественных элементов.

Ввести значения элементов массива. Затем нормировать элементы массива, разделив их на значение максимального по модулю элемента.

Вывести значения элементов измененного массива.

Для обработки и вывода элементов массива применить методы **`Array.ConvertAll()`** и **`Array.ForEach()`**. Способы обработки определять с помощью лямбда-выражений.

```
public static TOutput[] ConvertAll<TInput, TOutput>
  (TInput[] array, Converter<TInput, TOutput> converter )
public delegate TOutput Converter<in TInput, out TOutput>
(TInput input)
public static void ForEach<T>(T[] array, Action<T> action )
public delegate void Action<in T>(T obj)
```

In (Generic Modifier) [https://msdn.microsoft.com/en-us/library/dd469484.aspx]
Out (Generic Modifier) [https://msdn.microsoft.com/en-us/library/dd469487.aspx]

Задача 5. Лямбда-выражения как аргументы метода Array.Sort( )

Код метода Main()

```
int[] ar = { 6, 5, 4, 4, 3, 2, 4, 1, 2, 3, 4, 2, 4 };
Array.Sort(ar,  // сортировка по убыванию:
            (int x, int y) => {
                if (x < y) return 1;  // нарушен порядок
                if (x == y) return 0;
                return -1;
            }
          );
foreach (int memb in ar)
    Console.Write("{0} ", memb);
Console.WriteLine();
Array.Sort(ar,  // сортировка по четности:
            (x, y) =>   // явный тип параметров не обязателен
            {
                        if (x % 2 != 0 & y % 2 == 0) return 1;
                        if (x == y) return 0;
                        return -1;  // верный порядок
            }
          );
foreach (int memb in ar)
     Console.Write("{0} ", memb);
```

Результаты выполнения:

6 5 4 4 4 4 4 3 3 2 2 2 1

2 2 2 4 4 4 4 4 6 1 3 3 5

Задача 6

•Создать класс **Plant**:

- закрытые вещественные поля: рост (**growth**), светочувствительность (**photosensitivity**) и морозоустойчивость (**frostresistance**). Светочувствительность и морозоустойчивость – параметры имеющие значение от 0 до 100 включительно.

- открытый конструктор с тремя параметрами;

- открытые свойства для информации о росте, светочувствительносте и морозоустойчивости;

- переопределенный метод **ToString()**, выводящий информацию о растении на экран (рост, светочувствительность, морозоустойчивость).

В основной программе пользователь с клавиатуры вводит число **n**. Создайте массив из **n** растений и заполните их объектами, инициализированными случайными вещественными числами: рост от 25 до 100 см, светочувствительность от 0 до 100, морозоустойчивость 0 до 80.

•Выведите на экран, используя метод **Array.ForEach()**, сведения о растениях из массива.

•Отсортируйте массив с использованием метода **Array.Sort()** по убыванию роста и снова выведите. Используйте анонимный метод.

•Затем отсортируйте массив с использованием метода **Array.Sort()** по возрастанию морозоустойчивости и выведите на экран. Используйте лямбда-выражение.

•Затем отсортируйте массив с использованием метода **Array.Sort()** по четности светочувствительности и выведите на экран. Используйте самостоятельно определенный метод.

•Измените параметр морозоустойчивость у всех растений массива, используя метод **Array.ConvertAll()**. Морозоустойчивость меняем по следующему правилу: четные значения уменьшаем в 3 раза, нечетные значения уменьшаем в 2 раза.

Задания

1. В консольном приложении сформировать массив из десяти случайных целых элементов, со значениями из диапазона (–15; 15). Используя метод **Array.Sort()** отсортировать массив в порядке возрастания абсолютных значений его элементов, признак сортировки задавать лямбда-выражением. Исходный и отсортированный массивы вывести на экран. Для представления каждого значения элемента массива отвести четыре позиции.
2. В консольном приложении сформировать массив **A** из десяти случайных целых элементов, со значениями из диапазона (0; 20). Используя метод **Array.ConvertAll()** получить массив **B** вещественных значений, каждое из которых представляет собой значение функции **1/*x*** в точках, заданных элементами массива **A**, преобразование задавать лямбда-выражением. Значения элементов **A** и **B** вывести на экран. Точность вывода вещественных значений: два знака после десятичного разделителя.
3. В консольном приложении сформировать массив **A** из десяти случайных вещественных элементов, со значениями из диапазона (-3; 3). Используя метод **Array.ConvertAll()** получить массив **B** целых значений, каждое из которых представляет собой целые части для неотрицательных, и 0 для отрицательных элементов массива **A**, преобразование задавать анонимным методом. Значения элементов **A** и **B** вывести на экран. Точность вывода вещественных значений: два знака после десятичного разделителя.