Основы программирования

Иерархия классов

Стандартные интерфейсы

Стандартные интерфейсы

Реализация стандартных интерфейсов позволяет пользовательским классам участвовать в стандартных механизмах и схемах наравне со стандартными классами.

Мы рассмотрим:

- IComparable;
- IComparer;
- IClonable.

Сортировка массива

Пример 1

```
int [] mas = new int [4] {3, 4, 9, 6};
Array.Sort(mas);
```

Пример 2

```
class Books
{int price; string author, name;}

Books [] mas = new Books [4];
/* 200   Павловская   Программирование C#
   400   Шилд         Справочник C#
   500   Троэлсен     C#                    */
Array.Sort(mas);
```

Интерфейс IComparable

Реализуется когда объекты класса нужно уметь сравнивать между собой на «больше» и «меньше». Используется при сортировке массива.

- Пространство имен:
  `System.Collections`
- Содержит только один метод:
  `int CompareTo(object o)`

Метод `Sort` по умолчанию работает только для наборов примитивных типов, как `int` или `string`. Для сортировки наборов сложных объектов применяется интерфейс `IComparable`.

Интерфейс IComparable. Возвращаемое значение

- «1» - если объект, у которого вызывается метод, «больше» чем объект, переданный в качестве параметра;
- «-1» - если объект, у которого вызывается метод, «меньше» чем объект, переданный в качестве параметра;
- «0» - в том случае, если объекты «равны».

Отношения «больше», «меньше» и «равны» определяют, исходя из семантики (смысла) задачи.

Пример. Класс MyClass

```
class MyClass : IComparable
{
      int count;
      string description;

public MyClass(int c, string d)
      {
      count = c;
      description = d;
      }
```

Пример. Реализация CompareTo

```
public int CompareTo(object o)   {
MyClass mo = o as MyClass;

if (mo != null)
  if (this.count > mo.count) return 1;
  else
    if (this.count < mo.count) return -1;
    else
    return 0;
else
throw new Exception("Не сравнить");
}
}
```

Пример

```
…
MyClass m1 = new MyClass(1, "1 объект");
MyClass m2 = new MyClass(2, "2 объект");
if (m1.CompareTo(m2) > 0)
     Console.WriteLine("m1 > m2");
     else
     if (m1.CompareTo(m2) < 0)
     Console.WriteLine("m1 < m2");
     else Console.WriteLine("m1 = m2");
…
```

Пример. Пояснения

- В методе, унаследованном от интерфейса `IComparable`, определен критерий сравнения объектов на «больше» и «меньше».
- При сравнении объектов большим будет считаться тот объект, у которого значение целого поля `count` больше.

IComparable и сортировка массива

Реализация интерфейса `IComparable` позволяет использовать объекты производного класса в стандартных технологиях сравнения объектов.

```
MyClass[] mas = new MyClass[3];
mas[0] = new MyClass(5, "5");
mas[1] = new MyClass(1, "1");
mas[2] = new MyClass(2, "2");
Array.Sort(mas);
```

Как изменится код?

Как изменится следующий фрагмент кода чтобы сортировка производилась по:

- убыванию;
- букве;
- строке?

```
if (var1 > var2) return 1;
else
  if (var1 < var2) return -1;
  else
  return 0;
```

Пример 2. Класс Person

```
class Person : IComparable        {
  public string Name { get; set; }
  public int Age { get; set; }

  public int CompareTo(object o) {
  Person p = o as Person;
    if (p != null)
    return this.Name.CompareTo(p.Name);
    else
    throw new Exception("Сравнение невозможно");
}
}
```

Критерий сравнения – свойство `Name` объекта `Person`

Пример 2. Сортировка по строкам

```
Person p1 = new Person {Name="Глеб", Age=34};
Person p2 = new Person {Name="Игорь", Age =20};
Person p3 = new Person {Name="Ася", Age=17 };

Person[] people = new Person[] { p1, p2, p3 };
Array.Sort(people);



foreach(Person p in people) {
  Console.WriteLine("{0} - {1}", p.Name, p.Age);
}
```

Обобщенная версия IComparable

`IComparable` имеет обобщенную версию.

Можно сократить и упростить его применение в классе `Person`, используя необобщенную форму.

Посмотреть

https://metanit.com/sharp/tutorial/4.4.php

https://metanit.com/sharp/tutorial/3.12.php

Необобщенная форма IComparable

```
class Person : IComparable    {


public string Name { get; set; }
public int Age { get; set; }

public int CompareTo(object o) {
Person p = o as Person;
if (p != null)

return
this.Name.CompareTo(p.Name);

else
throw new Exception("Ошибка");
}
}
```

```
class Person : IComparable<Person>
        {

public string Name { get; set; }
public int Age { get; set; }

public int CompareTo(Person p) {



return
this.Name.CompareTo(p.Name);



}
}
```

Интерфейс IComparer

Предназначен для задания критерия сортировки
Позволяет описать различные правила сравнения объектов.

- Пространство имен:
  `System.Collections`
- Содержит только один метод:
  `int Compare (object o1, object j2)`

Интерфейс IComparar. Возвращаемое значение

- «1» - если первый параметр «больше» второго;
- «-1» - в том случае, если первый параметр «меньше» второго;
- «0» - в том случае, если объекты «равны».

Как и в методе `CompareTo`, отношения «больше», «меньше» и «равны» определяются, исходя из семантики задачи.

Порядок применения критерия сортировки

1. Описывается *вспомогательный класс*, реализующий интерфейс `IComparer` объект которого и будет являться критерием сравнения объектов пользовательского класса.
   Таких вспомогательных классов может быть несколько для реализации различных критериев сортировки.
2. Объект критерия передается в качестве параметра в стандартный метод сортировки массива пользовательских объектов.
   Для различных видов сортировки необходим свой объект критерия.

Пример. Класс MyClass

```
class MyClass   {
int count; string description;

public MyClass(int c, string d)  {
count = c; description = d;      }

public int Count{
get   {     return count;    }
set   {     count = value;   }     }

public string Description  {
get   {     return description;   }
set   {     description = value;  }
}}
```

Пример. Критерий 1 “SortByCountUp”

```
class SortByCountUp : IComparer  {

public int Compare(object o1, object o2)   {
MyClass mc1 = o1 as MyClass;
MyClass mc2 = o2 as MyClass;
…

if (mc1.Count > mc2.Count) return 1;
else
if (mc1.Count < mc2.Count) return -1;
else return 0;
}
}
```

Пример. Критерий 2 “SortByCountDown”

```
class SortByCountDown : IComparer {

public int Compare(object o1, object o2) {
MyClass mc1 = o1 as MyClass;
MyClass mc2 = o2 as MyClass;
…

if (mc1.Count < mc2.Count) return 1;
  else
  if (mc1.Count > mc2.Count) return -1;
    else return 0;
}
}
```

Пример. Критерий 3 “SortByDescriptionUp”

```
class SortByDescriptionUp : IComparer       {

public int Compare(object o1, object o2)    {
MyClass mc1 = o1 as MyClass;
MyClass mc2 = o2 as MyClass;
…

return mc1.Description.CompareTo(mc2.Description);
}
}
```

Пример. Критерий SortByDescriptionDown

```
class SortByDescriptionDown : IComparer {

public int Compare(object o1, object o2)  {
MyClass mc1 = o1 as MyClass;
MyClass mc2 = o2 as MyClass;
…

return mc2.Description.CompareTo(mc1.Description);
}
}
```

Пример. Проверка

```
MyClass[] mas = new MyClass[3];
mas[0] = new MyClass(5, "5");
mas[1] = new MyClass(1, "1");
mas[2] = new MyClass(7, "7");

Array.Sort(mas, new SortByCountUp());
Array.Sort(mas, new SortByCountDown());
Array.Sort(mas, new SortByDescriptionUp());
Array.Sort(mas, new SortByDescriptionDown());
```

Пример 2. Необобщенная форма

```
class PeopleComparer : IComparer<Person> {

public int Compare(Person p1, Person p2)
{
if (p1.Name.Length > p2.Name.Length)
    return 1;
else if (p1.Name.Length < p2.Name.Length)
    return -1;
else
    return 0;
}
}
```

Клонирование

Клонирование – это создание копии объекта.

Два вида клонирования объектов:

- поверхностное;
- глубокое.

Поверхностное клонирование

Создается точная копия объекта:

- все значащие поля копируются в клон,
- поля, являющиеся ссылками на другие объекты, также в точности копируются в клон – то есть остаются направленными на старые объекты, новых объектов для полей-ссылок поверхностное клонирование не создает.

Полезно, если у клонируемого объекта нет полей, ссылающихся на другие объекты.

Поверхностное клонирование реализовано в методе `MemberwiseClone`() в классе `Object`.

Глубокое клонирование

Реализуется если необходимо создание полного независимого глубокого клона объекта.

Алгоритм глубокого клонирования не специфицирован и описывается разработчиком.

При этом класс должен реализовывать интерфейс `ICloneable`, в котором описан единственный метод:

```
object Clone();
```

Глубокое клонирование

В методе `Clone()` и реализуется алгоритм глубокого клонирования.

Реализация классом интерфейса `ICloneable` говорит о том, что объекты класса умеют создавать свои глубокие клоны.

В заголовке метода `Clone()` в качестве возвращаемого результата указан тип `object`. Это означает, что при вызове операции получения глубокого клона необходимо явное преобразование полученного клона к искомому типу.

Рекомендации по клонированию …

Их следует основывать на иерархичном построении поверхностных (легких клонов):

- сложный объект сначала поверхностно клонируется на самом верхнем уровне, а затем
- внутренние ссылки направляются на поверхностные клоны внутренних ссылок, хранящихся в клонируемом объекте, и т.д.

Пример

```
class MyClass : ICloneable   {
int[] mas;
int length;

public MyClass(int l)      {
mas = new int[l];
length = mas.Length;       }

public int this[int i]     {
get    { return mas[i]; }
set    { mas[i] = value;   }    }

public int Length {
get    { return length;    }    }
```

Пример

```
public object LightClone()                {
return this.MemberwiseClone();            }

public object Clone()                  {
MyClass m = (MyClass)this.MemberwiseClone();
m.mas=(int[])mas.Clone();//вызов вернет глубокий клон
внутреннего объекта
return m;                 }
```

Пример

```
//вариант метода клонирования без
//использования поверхностных клонов

//     public object Clone()   {
//     MyClass m = new MyClass(length);
//     for (int i = 0; i < length; i++)     {
//     m[i] = mas[i];
//     }
//     return m;
//     }
}
```

Пример

```
MyClass o = new MyClass(3);
for (int i = 0; i < o.Length; i++)  {
   o[i] = i;
}
Console.WriteLine("Клонируемый объект:");

for (int i = 0; i < o.Length; i++) {
   Console.WriteLine(o[i]);
}
```

Пример

```
MyClass o1= (MyClass)o.LightClone();//поверхн.
o1[0] = 10;

Console.WriteLine("Клонируемый объект:");
for (int i = 0; i < o.Length; i++) {
   Console.WriteLine(o[i]);
}
Console.WriteLine("Клонированный объект:");
for (int i = 0; i < o1.Length; i++){
   Console.WriteLine(o1[i]);
}
```

Пример. Вывод на экран

Клонируемый объект:
0
1
2
Клонируемый объект:
10
1
2
Клонированный объект:
10
1
2

Пример. Пояснение

Как видно из примера, клонируемый объект `o` и его поверхностный клон `o1` ссылаются на один и тот же массив целых чисел.

Поэтому изменение состояния копии `o1` влечет за собой автоматическое изменение состояния исходного объекта `o`.

Пример. Продолжение

```
MyClass o = new MyClass(3);

for (int i = 0; i < o.Length; i++) {
     o[i] = i;
}

Console.WriteLine("Клонируемый объект:");
for (int i = 0; i < o.Length; i++) {
     Console.WriteLine(o[i]);
}
```

Пример. Продолжение

```
MyClass o2= (MyClass)o.Clone();  //создание
                                 глубокого клона
o2[0] = 10;
Console.WriteLine("Клонируемый объект:");
for (int i = 0; i < o.Length; i++){
Console.WriteLine(o[i]);}
Console.WriteLine("Клонированный объект:");
for (int i = 0; i < o2.Length; i++){
Console.WriteLine(o2[i]);}
```

Пример. Вывод на экран

Клонируемый объект:
0
1
2
Клонируемый объект:
0
1
2
Клонированный объект:
10
1
2