Концепция наследования и ее реализация в языке C#

Комментарий к слайду

В данной лекции будут рассмотрены вопросы, относящиеся к истории развития, идеологии, математическому основанию и обзору возможностей наследования – одной из фундаментальных концепций, на которых зиждется объектно-ориентированное программирование.

Содержание лекции

Комментарий к слайду

Коротко о содержании лекции.

В ходе лекции будут рассмотрены важнейшие научные исследования, относящиеся к эволюции подходов к математическому моделированию одной из важнейших для объектно-ориентированного подхода к программированию концепций, а именно, наследования.

Прежде всего, будет сформулировано определение наследования.

Затем будет представлен сравнительный анализ путей реализации концепции наследования в языках объектно-ориентированного программирования и в computer science.

При этом будут подробно исследованы особенности формализации концепции наследования посредством диаграмм Х.Хассе и фреймов Н.Руссопулоса.

Особое внимание будет уделено реализации механизмов наследования для объектов языка программирования C#, включая иерархическую организацию классов, а также множественное наследование.

Отображение классов Microsoft .NET в классы языков программирования SML и C# проиллюстрирует практику применения концепции наследования.

Лекция завершится обзором литературы для более глубокого исследования материала.

Основные результаты исследований в области наследования

1950-е – Х.Хассе (Helmut Hasse, 1898-1979) предложил диаграммы (позже названные в его честь) для графической иллюстрации отношения частичного порядка

1976 – Н.Руссопулос (N.D.Roussopulos) изобрел фреймы для моделирования предметных областей и ввел ISA-отношение частичного порядка

1979 – Д.Скотт (Dana S. Scott) сформулировал теорию полных и непрерывных решеток, которая используется в диаграммах потоков данных

1988-90 – Л.Карделли, У.Кук и др. (Luca Cardelli, William R. Cook et al.) исследовали (денотационную) семантику (множественного) наследования

Комментарий к слайду

Напомним ход эволюции теорий, лежащих в основе современного подхода к наследованию.

Еще в 50-х г.г. прошлого столетия Хельмут Хассе (Helmut Hasse, 1898-1979) предложил использовать диаграммы особого рода для графической интерпретации отношения частичного порядка. Заметим, что позднее диаграммы, открытые ученым, получили название в его автора и стали называться диаграммами Хассе. И по сей день диаграммы Хассе являются наиболее широко распространенной графической формализацией механизма наследования.

Затем, в 1976 году Н.Руссопулос (N.D.Roussopulos) впервые применил фреймовую нотацию для моделирования отношений между объектами тех или иных предметных областей. Ученым было также введено так называемое ISA-отношение частичного порядка, которое адекватно моделирует понятие наследования. Заметим, что обозначение ISA возникло от английских слов "… is a …"?, означающих «… является одним из … » и хорошо иллюстрирует суть понятия наследования на естественном языке.

Позднее, в 1979 году, Д.Скотт (Dana S. Scott) сформулировал теорию полных и непрерывных решеток, которая используется в диаграммах потоков данных. Решетка Д.Скотта представляет собой модель частично упорядоченного множества, или, иначе, модель иерархии классов.

Затем, в 1988-90 г.г. учеными Л.Карделли (Luca Cardelli), У.Куком (William R. Cook) и др. была исследована семантика наследования. При этом был построен вариант денотационной семантики, который, как оказалось, адекватно формализовал не только единичное, но и множественное наследование.

Наследование и методы его моделирования

Вообще говоря, под ***наследованием*** понимается свойство производного объекта сохранять поведение (атрибуты и операции) родительского.

Для языка программирования наследование означает, что (некоторые) свойства и методы базового класса равно применимы к его производным объектам (и их конкретизациям).

Наследование моделируется посредством отношения (иерархии) частичного порядка и адекватно отображается графически посредством:

1. Фреймовой нотации Н.Руссопулоса (N.D. Roussopulos);
2. Диаграм Хассе (H.Hasse)

Комментарий к слайду

Кратко обсудив историю развития концепции наследования и ее формализаций, перейдем к понятийному аппарату.

Под наследованием в дальнейшем будем понимать свойство производного объекта сохранять поведение родительского объекта. Под поведением будем иметь в виду для математического объекта его атрибуты и операции над ним, а для языкового объекта ООП – поля и методы.

Таким образом, применительно к языку программирования концепция наследования означает, что свойства и методы базового класса равно применимы к его производным объектам. Заметим, что дочерний объект не обязательно наследует все без исключения атрибуты и операции родительского, а лишь некоторые из них. Такой подход характерен к как для классов объектов в целом, так и для отдельных их конкретизаций, или, иначе, экземпляров.

Как уже упоминалось в ходе лекции, теоретическая концепция наследования удовлетворительно моделируется посредством отношения (или, точнее, иерархии) частичного порядка. Существует целый ряд формализаций наследования, но наиболее адекватными и концептуально ясными являются графические модели. Среди них следует выделить уже упомянутые ранее подходы: фреймовую нотацию Н.Руссопулоса и диаграмы Хассе.

Свойства отношения частичного порядка

1. Рефлексивность:
$\forall$ a: a ISA a

2. Транзитивность:
$\forall$ a,b,c: a ISA b, b ISA c $\Rightarrow$ a ISA c

3. Антисимметричность:
$\forall$ a,b : a ISA b $\Rightarrow$ NOT (b ISA a)

Комментарий к слайду

Отношение частичного порядка обладает следующими теоретически интересными и практически полезными свойствами.

Во-первых, оно является рефлексивным, т.е. любой объект языка программирования или формальной модели предметной области находится отношении частичного порядка с самим собой. Формальная запись свойства рефлексивности для отношения частичного порядка выглядит следующим образом:

$$\forall \text{ a: a ISA a.}$$

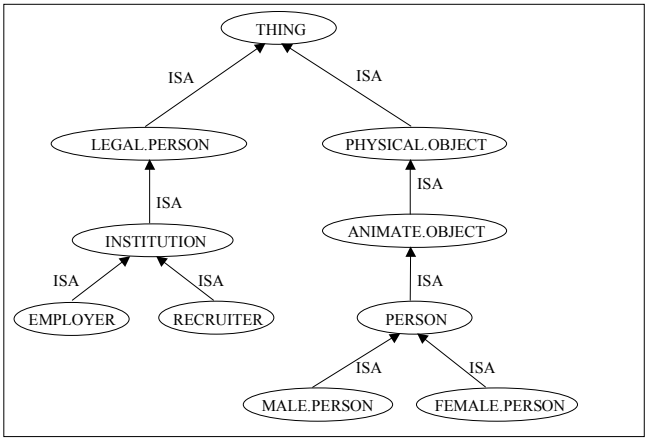
Другой важной особенностью отношения частичного порядка является транзитивность. Суть транзитивности состоит в том, что если существует цепочка из (трех) объектов, связанных отношением частичного порядка, то первый и последний элементы этой цепочки можно связать тем же отношением. Это свойство гарантирует построение на множестве графа отношения частичного порядка в форме «решетки». Формальная запись свойства транзитивности для отношения частичного порядка выглядит следующим образом:

$$\forall \text{ a,b,c: a ISA b, b ISA c} \Rightarrow \text{a ISA c.}$$

Наконец, еще одним фундаментальным свойством отношения частичного порядка как модели наследования является антисимметричность. Это свойство разрешает наследование только в одном направлении (и запрещает его в противоположном). Формальная запись свойства антисимметричности для отношения частичного порядка выглядит следующим образом:

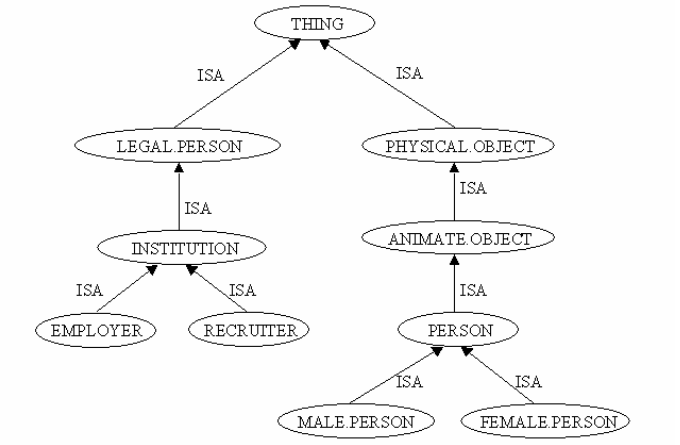
$$\forall \text{ a,b : a ISA b} \Rightarrow \text{NOT (b ISA a).}$$

Фрейм как пример моделирования наследования



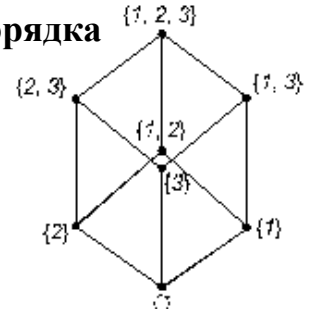
Комментарий к слайду

По завершении исследования свойств отношения частичного порядка, перейдем к рассмотрению формализаций, моделирующих наследование. Проиллюстрируем графическую интерпретацию отношения частичного порядка на примере фреймовой нотации Н. Руссопулоса. Рассмотрим следующий фрейм, который связывает отношением частичного порядка понятия «сущность» (THING), «юридическое лицо» (LEGAL.PERSON), «учреждение» (INSTITUTION), «работодатель» (EMPLOYER), «кадровое агентство» (RECRUITER), «физический объект» (PHYSICAL.OBJECT), «одушевленный объект» (ANIMATE.OBJECT), «человек» (PERSON), «мужчина» (MALE.PERSON) и «женщина» (FEMALE.PERSON).



Как видно из структуры фрейма, понятия (или, точнее, концепты) изображены в овалах. Направленные ISA-дуги соединяют понятия, образуя иерархию с направлением в сторону увеличения уровня общности (абстракции), например, от понятия «мужчина» к понятию «сущность». Рефлексивные и транзитивные дуги опущены для удобства восприятия; их без труда можно восстановить. Ориентированность дуг характеризует антисимметричность отношения частичного порядка: любая из дуг может иметь только одну стрелку со стороны увеличения уровня абстракции.

Использование диаграммы Хассе для моделирования частичного порядка



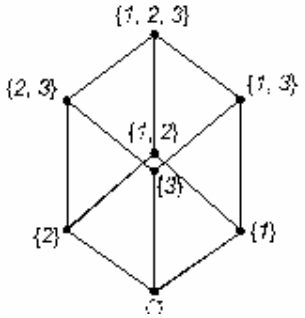
Данный пример иллюстрирует отношение IS IN между множествами {}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, и {1,2,3}.

Диаграмма Хассе состоит из точек, представляющих элементы множества, а также из соединяющих их линий, которые представляют собой отношения между элементами.

Комментарий к слайду

Другой продуктивной формализацией, моделирующей, в частности, наследование, является диаграмма Хассе. Проиллюстрируем использование диаграмм Хассе для графической интерпретации отношения между элементами класса или домена.

Диаграмма Хассе состоит из точек, которые представляют элементы множества (точнее, домена или класса), а также из соединяющих их линий, которые представляют собой отношения между элементами класса или домена (в данном случае интерпретируется отношение частичного порядка).



Данный пример иллюстрирует отношение IS IN («является подмножеством») между множествами {}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3} и {1,2,3}.

Заметим, что в случае графической интерпретации отношения частичного порядка с помощью диаграммы Хассе свойство антисимметричности рассматриваемого отношения было бы отображено в явном виде.

Пример единичного наследования на C# (1)

```
class A {     // базовый класс
 int a;
 public A() {...}
 public void F() {...}
}

class B : A { // подкласс (наследует свойства
класса A, расширяет класс A)
 int b;
 public B() {...}
 public void G() {...}
}
```


Комментарий к слайду

Исследовав основные свойства отношения наследования и способы его наглядной формализации, рассмотрим, каким образом эта теоретическая концепция реализуется в языках объектно-ориентированного программирования, и, в частности, в языке C#.

В простейшем случае язык программирования C# поддерживает единичное наследование. Проиллюстрируем реализацию механизма единичного наследования фрагментом программы на языке C#.

```
class A { // базовый класс
        int a;
        public A() {...}
        public void F() {...}
     }

class B : A {
// подкласс (наследует свойства класса A, расширяет класс A)
        int b;
        public B() {...}
        public void G() {...}
}
```

Применительно к языку программирования C# наследование есть отношение между классами. Данный пример содержит описание более общего (находящегося выше по ISA-иерархии) класса `A` и его подкласса – класса `B`. Заметим, что находящийся выше по ISA-иерархии класс принято называть базовым, а находящийся ниже – производным (или подклассом).

Подкласс `B` наследует свойства класса `A`, и, кроме того, возможно, расширяет его новыми (по сравнению с классом `A`) свойствами.

Пример единичного наследования на C# (2)

В наследует свойство a и метод F(), добавляя b и G():
- конструкторы не наследуются;
- наследуемые методы могут игнорироваться (см. далее)

Единичное наследование: подкласс может наследовать свойства единственного базового класса, однако может при этом реализовывать множественные интерфейсы.

Класс может наследовать свойства класса, но не структуры.

Структура не может наследовать свойства другого типа данных, однако может при этом реализовывать множественные интерфейсы.

Подкласс с неявным базовым классом наследует свойства класса объект (`object`).

Комментарий к слайду

Приведенный фрагмент программы на языке C# иллюстрирует простейший случай наследования, а именно, единичное наследование.

Рассмотрим подробнее, как именно производится реализация механизма наследования.

Производный класс `B` наследует свойство `a` и метод `F()` базового класса `A`. Кроме того, к свойствам производного класса `B` добавляется новое по сравнению с базовым классом `A` свойство `b`, а к методам – новый по сравнению с базовым классом `A` метод `G()`.

Заметим, что операторы конкретизации элементов базовых классов (которые в языке C# называют конструкторами) не наследуются производными классами.

Заметим также, что некоторые из наследуемых методов могут игнорироваться. В ходе единичного наследования производный подкласс может наследовать свойства единственного базового класса. Множественное наследование, демонстрирующее гибкость механизмов наследования в языке C#, реализуется на основе интерфейсов. Хотя производный класс в языке программирования C# может наследовать свойства базового класса, он не может наследовать свойств структуры. Реализация механизма наследования (в том числе множественного) для структур в языке программирования C# осуществляется (как и в случае с классами) посредством интерфейсов.

Вопросы множественного наследования посредством механизма интерфейсов будут более подробно обсуждены далее в ходе курса.

В случае неявного задания базового класса, производный класс наследует свойства наиболее общего класса Microsoft .NET, известного как объект (`object`) и аналогичного концепту THING («сущность») в рассмотренном ранее примере фреймовой нотации.

Множественное наследование в C#: интерфейсы (1)

```
public interface IList : ICollection, IEnumerable
{
     int Add (object value);
                                        // методы
     bool Contains (object value);
     ...
     bool IsReadOnly { get; }
                                        // свойство
     ...
     object this [int index] { get; set; }
                                        // индексатор
}
```


Комментарий к слайду

Как уже неоднократно упоминалось в ходе изложения, в языке программирования C# допускается не только единичное, но и множественное наследование.

Для реализации концепции множественного наследования в языке программирования C# предусмотрен так называемый механизм интерфейсов.

Для иллюстрации приведем фрагмент программы на языке C#, содержащей описание интерфейса:

```
public interface IList : ICollection, IEnumerable
     {
          int Add (object value);
               // методы
                    bool Contains (object value);
                    ...
                    bool IsReadOnly { get; }
               // свойство
                    ...
                    object this [int index] { get; set; }
               // индексатор
     }
```

Из приведенного примера можно видеть, что фрагмент программы на языке C# представляет собой описание общедоступного интерфейса `IList`, наследующего в иерархии элементы интерфейсов `ICollection` и `IEnumerable`, а также содержащего методы `Add` и `Contains`, свойство `IsReadOnly` и объект-индексатор `this`.

Множественное наследование в C#: интерфейсы (2)

Интерфейсом называется чисто абстрактный класс (с полиморфизмом), содержащий только описание без реализации.

- Интерфейс может содержать методы, свойства, индексаторы и события (не может содержать полей, констант, конструкторов, деструкторов, операторов и др.).
- Элементы `interface` неявно являются `public abstract` (`virtual`).
- Элементы `interface` не должны быть статическими.
- Классы и структуры могут реализовывать множественные интерфейсы.
- Интерфейс может расширять другой интерфейс.

Комментарий к слайду

После визуального знакомства с кодом на языке программирования C#, описывающим интерфейс, приведем более общее словесное определение этого механизма, имеющего принципиальное значение.

Интерфейсом называется чисто абстрактный класс (с поддержкой полиморфизма) содержащей только описания без реализации.

Концепция множественного наследования предполагает возможность наследования одним концептом языка свойств сразу нескольких языковых концептов. При этом в языке программирования C# принципиально допустимо множественное наследование, но область действия его ограничена интерфейсами. Множественное наследование классов в языке программирования C# недопустимо, однако неявно реализуемо посредством интерфейсов. Так, классы и структуры языка программирования C# могут реализовывать множественные интерфейсы.

Интерфейсы могут содержать в своем составе методы, свойства, индексаторы и события. Однако интерфейсы не могут содержать в своем составе полей, констант, конструкторов, деструкторов, операторов, а также вложенных типов.

Элементы интерфейса неявно являются виртуальными (т.е. общедоступными и абстрактными) объектами и описываются в языке C# как `public abstract` (или `virtual`).

Интерфейсы, будучи принципиально полиморфными (т.е. динамически означиваемыми) объектами, не могут содержать в своем составе статических элементов.

Наконец, свойства одного интерфейса могут быть расширены посредством другого интерфейса.

Множественное наследование в C#: интерфейсы (3)

- Класс может наследовать свойства единственного базового класса, но реализовывать множественные интерфейсы.
- Структура не может наследовать свойства типа, но может реализовывать множественные интерфейсы.
- Любой элемент интерфейса (метод, свойство, индексатор) должен быть реализован в базовом классе или унаследован от него.
- Реализованные методы интерфейса могут описываться как виртуальные (`virtual`) или абстрактные (`abstract`), т.е. интерфейс может быть реализован посредством абстрактного класса, но не как `override`.

Комментарий к слайду

Рассмотрим более подробно важнейшие свойства интерфейсов как реализации концепции множественного наследования в языке программирования C#.

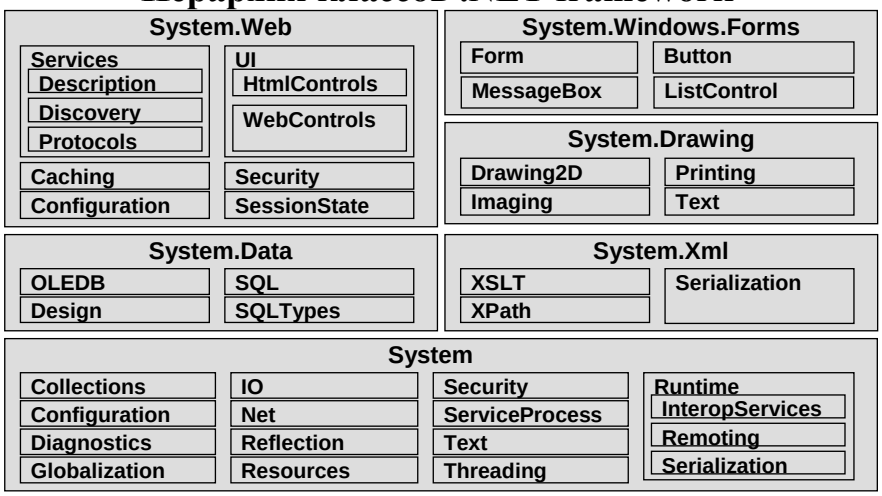
Как уже отмечалось, концепция множественного наследования в языке программирования C# в явной форме неприменима для классов. В случае классов существует возможность только единичного наследования, т.е. производный класс может наследовать свойства единственного базового класса. Механизм интерфейсов, таким образом, является неявной возможностью реализации концепции множественного наследования для языка программирования C#, поскольку для классов в языке допустима реализация множественных интерфейсов.

В отношении наследования структур в языке программирования C# наблюдается определенное сходство с классами. В частности, несмотря на то обстоятельство, что структуры не имеют возможности наследовать свойства типов, для них, также как и для классов, допустима реализация множественных интерфейсов.

При этом произвольный элемент интерфейса, будь то метод, свойство или индексатор, должен непременно быть либо реализован непосредственно в базовом классе, либо унаследован от него.

Кроме того, заметим, что реализованные в языке программирования C# методы интерфейса могут быть описаны либо как виртуальные (`virtual`), либо как абстрактные (`abstract`). Таким образом, интерфейс может быть реализован посредством абстрактного класса. В то же время не допускается реализация переопределенных интерфейсов по аналогии с переопределенными в производных классах методами, которые описываются посредством ключевого слова `override` языка программирования C#.

Иерархия классов .NET framework



Комментарий к слайду

Концепция наследования, рассмотренная нами применительно к объектно-ориентированному подходу к программированию, вполне может быть распространена и на случай систем и сред, поддерживающих проектирование и реализацию программного обеспечения.

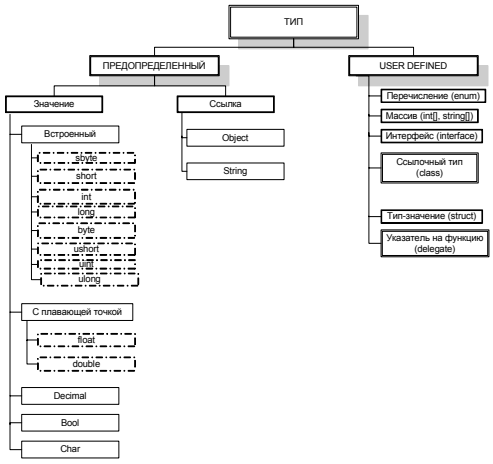
В этой связи не является исключением и среда интегрированной разработки приложений Microsoft .NET, классы в которой не имеют существенных различий с классами языка программирования C#.

Как видно из схемы, описывающей основные пространства имен Microsoft .NET Framework, организация их является вполне иерархической и реализует классическую схему единичного наследования.

Наиболее общим концептом иерархии является пространство имен System, характеризующее конфигурацию среды Microsoft .NET Framework и содержащее, в частности, параметры среды времени выполнения приложений, удаленной обработки данных, процессов, безопасности, ввода-вывода, системной конфигурации и др.

Среди подпространств пространство имен System можно выделить такие пространства имен, как System.Web, System.Windows.Forms, System.Data, System.Drawing и System.Xml, которые описывают такие характеристики среды Microsoft .NET Framework, как конфигурации веб-сервисов, формы ввода-вывода данных, форматы представления данных, графических подсистем и др.

Иерархия типов языка C# (фрагмент)



Комментарий к слайду

Данная схема иллюстрирует перечень классов Common Type System среды проектирования и реализации программного обеспечения Microsoft .NET применительно к языку программирования C#.

В частности, все многообразие типов можно разделить на предопределенные (заранее заданные системой программирования) и определенные пользователем (user defined).

К последним относятся перечисления, массивы, классы, интерфейсы и делегаты (указатели на функцию).

Предопределенные типы делятся на ссылочные типы (объекты и символьные строки) и типы-значения (встроенные – короткие и длинные целые со знаком и без знака, а также числа с плавающей точкой – с одинарной и двойной точностью).

Элементарные типы языка C# (в сопоставлении с языком SML)

C#	CTS	SML	Диапазон
sbyte	System.SByte	---	-128 .. 127
byte	System.Byte	byte	0 .. 255
short	System.Int16	int	-32768 .. 32767
ushort	System.UInt16	word	0 .. 65535
long	System.Int64	---	-2^{63} .. $2^{63}-1$
float	System.Single	real	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	---	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	---	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	bool	true, false
char	System.Char	char	Символ (в коде unicode)

Комментарий к слайду

Перейдем к более подробному рассмотрению основных типов, входящих в состав Common Type System (CTS) Microsoft .NET. При этом прикладное исследование языков программирования произведем в форме сопоставления отображений фрагментов систем типизации языков программирования SML и C# в систему типов CTS.

Для обеспечения большей наглядности сопоставления сведем отображения в следующую таблицу:

C#	CTS	SML	Диапазон
sbyte	System.SByte	---	-128 .. 127
byte	System.Byte	byte	0 .. 255
short	System.Int16	int	-32768 .. 32767
ushort	System.UInt16	word	0 .. 65535
long	System.Int64	---	-263 .. 263-1
float	System.Single	real	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	---	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	---	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	bool	true, false
char	System.Char	char	Символ (в коде unicode)

Даже из предварительного анализа таблицы видно, что система типизации языка программирования C# значительно богаче по сравнению с языком программирования SML. Можно заметить, что всякому типу языка программирования SML соответствует некоторый тип языка программирования C#, и их названия зачастую совпадают или являются подобными друг другу.

Наконец, отметим, что все без исключения типы обоих языков программирования однозначно отображаются в систему типизации Microsoft .NET, верхним иерархическим элементом которой является пространство имен System.

Преимущества наследования

1. Отсутствие необходимости явного указания свойств и методов для производных объектов
2. Гибкое определение уровня абстракции предметной области
3. Возможность моделирования сколь угодно сложной предметной области
4. Простой и прямолинейный подход к построению (производных) классов по заданной совокупности свойств и методов

Комментарий к слайду

Подводя итоги обсуждения наследования, одной из основополагающих концепций объектно-ориентированного подхода к программированию, кратко резюмируем положительные стороны явления и вытекающие из них практические возможности.

Прежде всего, концепция наследования устраняет необходимость многократного явного указания свойств и методов для производных объектов. Одного взгляда на схему пространств имен Microsoft .NET Framework достаточно, чтобы понять значимость этого свойства. Заметим, что на схеме показана лишь часть верхних «этажей» многоуровневой иерархии.

Кроме того, следует отметить, что наследование открывает возможности для гибкого определения уровня абстракции предметной области. Можно оперировать единственным, достаточно абстрактным, концептом System или THING, и, спускаясь по ISA-иерархии, вести рассуждения в терминах все более конкретных концептов и сущностей (или, в терминологии языка программирования C#, классов и объектов).

Таким образом, концепция наследования предоставляет возможность моделирования сколь угодно сложной предметной области посредством ISA-иерархии концептов (или классов языка программирования C#), моделирующих объекты реального мира.

Итак, наследование предоставляет простой и прямолинейный подход к построению классов (которые в терминологии языка программирования C# принято называть производными) по заданной совокупности свойств и методов так называемых базовых классов.

Заметим, что в языке программирования C# допустимо наследование свойств и методов как от единственного класса (единичное), так и от нескольких классов (множественное). Последний вид наследования реализуется посредством механизма интерфейсов.

Для моделирования наследования возможно воспользоваться фреймами Н.Руссопулоса и диаграммами Х.Хассе.

Библиография (1)

1. Cardelli L. A semantics of multiple inheritance. In: Information and Computation, vol.76, 1988, p.p. 138-164
2. Cook W., Palsberg J. A denotational semantics of inheritance and its correctness. In: OOPSLA 1989 as SIGPLAN, vol.24, No.10, Oct. 1989, p.p. 433-444
3. Cook W., Hill W.L., Canning P.S. Inheritance is not subtyping. In: Proc. 17th ACM Symposium on Principles of Programming Langauges, Jan. 1990, p.p. 125-135
4. Scott D.S. The lattice of flow diagrams.- Lecture Notes in Mathematics, 188, Symposium on Mathematics of Algorithmic Languages.- Springer-Verlag, 1971, p.p. 311-372

Комментарий к слайду

К сожалению, в рамках времени, отведенных на одну лекцию, можно лишь в общих чертах охарактеризовать специфику такой фундаментальной концепции для каждого языка объектно-ориентированного программирования и для подхода в целом, как наследование. Для более детального ознакомления с особенностями, достижениями и проблемами в теории моделирования этой концепции и практики реализации связанных с ней механизмов рекомендуется следующий список литературы:

1.Cardelli L. A semantics of multiple inheritance. In: Information and Computation, vol.76, 1988, p.p. 138-164

2.Cook W., Palsberg J. A denotational semantics of inheritance and its correctness. In: OOPSLA 1989 as SIGPLAN, vol.24, No.10, Oct. 1989, p.p. 433-444

3.Cook W., Hill W.L., Canning P.S. Inheritance is not subtyping. In: Proc. 17th ACM Symposium on Principles of Programming Langauges, Jan. 1990, p.p. 125-135

4.Scott D.S. The lattice of flow diagrams.- Lecture Notes in Mathematics, 188, Symposium on Mathematics of Algorithmic Languages.- Springer-Verlag, 1971, p.p. 311-372

Кратко остановимся на источниках. В работе [1] исследуется семантика множественного наследования. В работе [2] обсуждается применимость денотационной семантики для моделирования наследования, в частности, с целью обеспечения корректности. Работа [3] устанавливает связь между наследованием и приписыванием подтипов. В работе [4] представлено исчерпывающее описание теории решеток – продуктивной формализации для моделирования наследования.

Библиография (2)

5. Scott D.S. Identity and existence in intuitionistic logic.- In: Application of Sheaves.- Berlin: Springer, 1979, p.p. 600-696

6. Roussopulos N.D. A semantic network model of data bases, Toronto Univ., 1976

7. Frei G. Helmut Hasse (1898-1979), Expositiones Mathematicae 3 (1) (1985), 55-69

Комментарий к слайду

Продолжим обсуждение работ, посвященных исследованию концепции наследования в объектно-ориентированном подходе к программированию.

5.Scott D.S. Identity and existence in intuitionistic logic.- In: Application of Sheaves.- Berlin: Springer, 1979, p.p. 600-696

6.Roussopulos N.D. A semantic network model of data bases, Toronto Univ., 1976

7.Frei G. Helmut Hasse (1898-1979), Expositiones Mathematicae 3 (1) (1985), 55-69

Работа [5] представляет собой систематизацию важнейших понятий – тождества и существования – в логиках высших порядков и затрагивает вопросы принципиальной формализуемости реального мира. В работе [6] обсуждаются способы построения математически и логически корректных взаимоотношений понятий предметной области, т.е. строится своего рода объектная модель наследования, графически интерпретируемая фреймами Н.Руссопулоса. В работе [7] систематизируются научные взгляды Х.Хассе, создателя диаграмм Хассе – графической формализации, моделирующей наследование.