Основные понятия ООП: объекты, классы и методы

Комментарий к слайду

В данной лекции будут рассмотрены вопросы, относящиеся к идеологии, методологии и практике моделирования основных элементов объектно-ориентированного подхода к программированию посредством двухуровневой концептуализации. Особенности практической реализации основных аспектов концепции ООП излагаются на примере языка программирования C#.

Содержание лекции

Комментарий к слайду

Коротко о содержании лекции.

Прежде всего, необходимо составить предварительное представление об основных элементах ООП (объектах, классах и методах) на интуитивном уровне.

Затем речь пойдет о соотношении понятий объекта и класса и особенностях каждого из этих элементов ООП.

В качестве теоретической формализации понятий «класс» и «объект» будет предложена так называемая концептуальная модель.

Особенности реализации основных аспектов ООП применительно к Microsoft .NET будут проиллюстрированы такими конструкциями языка программирования C# как объекты, свойства и методы. Значительное внимание будет уделено сравнительному исследованию классов и структур языка программирования C#.

Далее в лекции слушателям будет предложен подход к созданию, инициализации и удалению объектов при помощи конструкторов и деструкторов языка программирования C#. По результатам исследования будут выявлены преимущества и недостатки объектного подхода к программированию.

Наконец, для желающих глубже исследовать предмет будут представлены ссылки на важнейшие работы теоретического и практического плана по теме лекции.

Основные работы в сфере моделирования ООП (1)

1924 – М.Шейнфинкель (Moses Shönfinkel) разработал теорию «простых» функций

1934 – А.Черч (Alonso Church) изобрел ламбда-исчисление и применил его в исследовании теории множеств

1971 – Д.Скотт (Dana S. Scott) предложил использовать полные и непрерывные решетки для формализации семантики (типизированного) ламбда-исчисления

Комментарий к слайду

Напомним ход эволюции теорий, лежащих в основе современных подходов к моделированию ООП.

Еще в 1924 г. М. Шенфинкель (Moses Shönfinkel) разработал простую (simple) теорию функций, которая фактически являлась исчислением объектов-функций и предвосхитила появление ламбда-исчисления.

Затем в 1934 г. А. Черч (Alonso Church) предложил исчисление ламбда-конверсий или ламбда-исчисление и применил его для исследования теории множеств. Вклад ученого был настолько фундаментальным, что теория до сих пор называется ламбда-исчислением и часто именуется в литературе ламбда-исчислением Черча. Заметим, что исчисление ламбда-конверсий адекватно формализует и объектно-ориентированный подход к программированию.

Позднее, в конце 60-х г.г., Д. Скоттом (Dana S. Scott) было предложено использовать для формализации семантики математических теорий так называемые домены (множества, адекватно формализующие рекурсивные вычисления). При этом на основе доменов Д. Скоттом был предложен так называемый денотационный подход к семантике. Такой подход предполагает анализ синтаксически корректных конструкций языка (или, иначе, денотатов) с точки зрения возможности вычисления их значений посредством специализированных функций. Важной особенностью формализации являлось использование полных и непрерывных решеток (которые моделируют отношение частичного порядка, т.е. иерархию классов в объектно-ориентированном подходе к программированию).

Основные работы в сфере моделирования ООП (2)

1980-е – Д.Скотт (Dana S. Scott) и М.Фурман (Michael P. Fourman) исследовали аппарат определенных дескрипций как средство формализации определений

1990-е – В.Э.Вольфенгаген (Vyatcheslav E. Wolfengagen) предложил схему двухуровневой концептуализации для моделирования объектов предметных областей (и языков программирования); данная модель адекватна как для объектов данных (ОД), так и для объектов метаданных (ОМД)

Комментарий к слайду

Затем, в 80-е г.г. автором уже известной нам из предыдущего курса теории вычислений Д.Скоттом (Dana S. Scott) вместе с М.Фурманом (Michael P. Fourman) были проведены исследования аппарата определенных дескрипций, который является адекватным средством для формализации определений. В нашем курсе мы будем использовать именно этот аппарат для математически строгой сокращенной записи определений объектов, типов и классов.

Позднее, в 90-е г.г. Вячеславом Вольфенгагеном (Vyatcheslav E. Wolfengagen) была создана так называемая двухуровневая схема концептуализации, основанная на двукратном применении постулата свертывания (до известной степени аналогичного операции ламбда-абстракции).

Заметим, что последняя модель позволяет вполне адекватно и непротиворечиво описывать объекты предметных областей с учетом их рассмотрения как в динамике, так и в статике. При этом двухуровневая схема концептуализации позволяет моделировать как объекты той или иной предметной области, так и объекты языков программирования.

Еще одним существенным преимуществом данной модели является возможность ее использования как применительно к объектам данных (ОД), так и применительно к объектам метаданных (ОМД).

Интуитивные определения основных понятий

Объектом называется математическое представление сущности реального мира (или предметной области), которое используется для моделирования.

Классом называется весьма общая сущность, которая может быть определена как совокупность элементов.

Свойством (или атрибутом) называется пропозициональная функция, определенная на произвольном типе (данных).

Методом (или функцией) называется операция, определенная над объектами некоторого класса.

Комментарий к слайду

Перейдем непосредственно к понятийному аппарату объектно-ориентированного подхода к программированию.

Прежде, чем будут изложены математически строгие определения (в том числе в терминах теории дескрипций), попытаемся сформировать предварительное представление об основных сущностях данного подхода.

Сформулируем на интуитивном уровне определения таких основополагающих для объектно-ориентированного подхода к программированию понятий, как объект, класс, свойство и метод.

Под объектом будем понимать математическое представление сущности реального мира (или предметной области), которое используется для моделирования.

Классом будем называть весьма общую сущность, которая может быть определена как совокупность элементов (нужно заметить, что класс при объектно-ориентированном подходе к программированию – это, как правило, первичное, неопределяемое понятие, до некоторой степени аналогичное теоретико-математическому понятию множества, или, точнее, домена).

Под свойством (или атрибутом) будем понимать пропозициональную функцию, определенную на произвольном типе (данных).

Методом (или функцией) назовем операцию, которая определена над объектами того или иного класса.

Соотношение понятий объекта и класса

Понятие класса является более общим, чем понятие объекта.

Объект является экземпляром класса.

Класс может рассматриваться как совокупность объектов (подобно тому, как множество есть совокупность элементов).

Класс может быть элементарным или подразделяться на подклассы (подобно тому как множество подразделяется на подмножества).

Например, класс **PERSON** содержит подкласс **STUDENT**, который, в свою очередь, содержит объект **John_Smith**.

Комментарий к слайду

Первичным понятием объектно-ориентированного подхода к программированию является понятие класса как совокупности объектов.

Заметим, что класс при объектно-ориентированном подходе является аналогом понятия типа в том смысле, что к нему относят лишь объекты, отобранные по определенному правилу. Это правило возможно формализовать математически посредством предикатной функции, т.е. функции, область значений которой совпадает со значениями истинности: «истина» и «ложь». При этом, тот или иной объект относится к классу, если значение аппликации функции к данному объекту истинно, и не относится в противном случае.

Функцию такого рода принято называть индивидуализирующей функцией. Индивидуализирующая функция фактически является моделью экспертной классификации.

Важным условием при исследовании объектно-ориентированного подхода к программированию является установление взаимосвязей фундаментальных сущностей. Отметим в этой связи, что понятие класса является изначально более общим, чем понятие объекта. Точнее говорят, что объект является экземпляром (instantiation) класса.

Таким образом, класс может рассматриваться как совокупность объектов (подобно тому, как множество, а точнее говоря, домен, есть совокупность элементов).

В рамках объектно-ориентированного подхода к программированию произвольный класс может быть элементарным либо подразделяться на подклассы (подобно тому как множество или домен подразделяется на подмножества или субдомены).

Например, более общий класс **PERSON** может содержать внутренний подкласс **STUDENT**, который, в свою очередь, содержит конкретный объект **John_Smith**.

Концептуализация как модель объекта

1. Модель основана на типизированном λ-исчислении, семантика которого моделируется посредством полных и непрерывных решеток Д.Скотта.

2. Вводятся аппликативные структуры с приписыванием типов.

3. Для формализации определений используются определенные дескрипции Скотта-Фурмана вида:

IxФ, что означает “тот единственный x, для которого Ф истинно”.

4. Произвольный объект моделируется тройкой

<концепт, индивид, состояние>,

составляющие которой соединены соотнесениями.

Комментарий к слайду

Как уже отмечалось, в 90-е г.г. Вячеславом Вольфенгагеном (Vyatcheslav E. Wolfengagen) была создана так называемая двухуровневая схема концептуализации, основанная на двукратном применении постулата свертывания, до известной степени аналогичного операции ламбда-абстракции.

Рассмотрим более подробно основные аспекты данной формализации объектно-ориентированного подхода к программированию. В общих чертах, схема построения модели выглядит следующим образом.

Основу модели составляет типизированный вариант ламбда-исчисления, семантика которого моделируется посредством полных и непрерывных решеток Д.Скотта.

Для описания объектов произвольной сложности вводятся аппликативные структуры с приписыванием типов. Последнее обстоятельство необходимо для формализации иерархии классов, в которые объединяются объекты.

Для формализации определений используются определенные дескрипции Скотта-Фурмана вида:

I x Ф,

что означает «тот единственный объект x, для которого значение индивидуализирующей функции Ф истинно».

При таком подходе произвольный объект моделируется упорядоченной тройкой элементов вида

<концепт, индивид, состояние>,

составляющие которой соединены соотнесениями.

Принцип концептуализации

$$||\mathbf{I}x\Phi(x)||i = d \Leftrightarrow \{d\} = \{d \in D \mid ||\Phi(\text{đ})||i = true\}$$

Значением индивидного концепта является функция из соотнесений в индивиды

Произвольный объект d предметной области D может быть единственным образом индивидуализирован посредством функции Ф (где i означает соотнесение).

D можно также интерпретировать как класс, а d – как объект языка программирования.

Комментарий к слайду

Основным соотношением формальной схемы двухуровневой концептуализации является так называемый принцип концептуализации, до известной степени аналогичный принципам свертывания, которые (в той или иной форме) присутствуют практически во всех математических теориях (в частности, в исследованных ранее в рамках курса ламбда-исчислении и комбинаторной логике).

Формальная запись принципа концептуализации посредством определенных дескрипций выглядит следующим образом:

$$||\mathbf{I}x(x)\Phi(x)||i = d \Leftrightarrow \{d\} = \{d \in D \mid ||\Phi(\text{đ})||i = true\}$$

Приведем словесную интерпретацию формулы:

Значением индивидного концепта
является функция
из соотнесений в индивиды

С точки зрения рассматриваемой формальной теории, произвольный объект d предметной области D может быть единственным образом индивидуализирован посредством функции Ф (где i означает соотнесение).

Заметим, что при соотнесении общей теории с языками программирования домен D можно также интерпретировать как класс, а элемент домена d – как объект языка программирования.

Классы в C#

- Ссылочный тип, определенный пользователем (аналогично языкам C++ и Java)
- Единичное наследование классов
- Множественное наследование интерфейсов
- Члены (элементы) класса:
  - константа, поле, метод, оператор, конструктор, деструктор;
  - свойство, индексатор, событие;
  - статические и инициализированные члены.
- Доступ к членам класса (`public, protected, private`(по умолч.),`internal,protected internal`)
- Инициализация – посредством оператора `new`

Комментарий к слайду

Рассмотрев интуитивное определение понятия класса, а также представив домены как формальную модель классов языков программирования в целом, остановимся более подробно на классах в языке объектно-ориентированного программирования C#.

По аналогии с другими известными языками объектно-ориентированного программирования (в частности, C++ и Java), под классом в языке C# понимается ни что иное, как ссылочный тип, определенный пользователем.

При этом для классов языка программирования C# допустимо только единичное наследование. В случае необходимости реализации множественного наследования, наследование возможно посредством механизма интерфейсов, который будет подробнее рассмотрен далее в ходе курса.

Членами (или, иначе, элементами) класса языка программирования C# могут являться следующие конструкции:
- константа, поле, метод, оператор, конструктор, деструктор;
- свойство, индексатор, событие;
- статические и инициализированные члены.

Доступ к членам класса определяется исходя из значения модификатора области действия идентификатора класса, который может принимать следующие значения: `public, protected, private` (данное значение используется по умолчанию),`internal,protected internal.`

Инициализация объекта класса языка программирования C# производится посредством оператора `new`, который будет рассмотрен в ходе данной лекции.

Описание и использование классов в C# (1)

```
class C {
  …
  int value = 0;
  …
  }
```

Инициализация поля (значения) факультативна и не должна давать доступа к полям и методам того же типа

Доступ внутри класса C: `... value ...`

Доступ из других классов:
```
C c = new C();
... c.value ...
```


Комментарий к слайду

Рассмотрим манипулирование классами на примере следующих фрагментов программ на языке C#.

Прежде всего, приведем простейшее описание класса. Описание класса `C` с целочисленным полем `value` на языке C# имеет вид:

```
class C {
        … int value = 0; …
}
```

Заметим, что в описании класса `C` на языке программирования C# кроме рассмотренного поля `value` могут присутствовать и другие поля (т.е. атрибуты объектов класса) допустимых в языке C# типов, а также методы (т.е. способы манипулирования объектами данного класса).

В языке программирования C# инициализация поля (т.е. связывание его с начальным значением) не является необходимым требованием. Для обеспечения безопасности программного кода и в силу реализации принципа инкапсуляции, инициализация поля некоторого класса `C` не должна открывать возможностей для доступа к полям и методам этого типа. При этом доступ к элементам класса внутри этого класса реализуется посредством обращения и не требует полного имени объекта:

```
        ... value ...
```

В отличие от предыдущего случая, доступ из сторонних классов требует указания полного имени объекта (в примере последовательно производятся инициализация и обращение):

```
        C c = new C();
        ... c.value ...
```

Описание и использование классов в C# (2)

Описание:

```
class Rectangle {
   Point origin;
   public int width, height;
   public Rectangle()
   {origin = new Point(0,0); width=height=0;}
      public Rectangle (Point p, int w, int h)
        {origin = p; width = w; height = h;}
      public void MoveTo (Point p) {origin = p;}
   }
```

Применение:

```
Rectangle r=new Rectangle(new Point(10,20),5,5);
int area = r.width * r.height;
r.MoveTo(new Point(3, 3));
```


Комментарий к слайду

Рассмотрим более развернутый пример описания классов и манипулирования их элементами.

Приведем описание класса `Rectangle`, моделирующего прямоугольник с полями `origin`, `width`, и `height`, моделирующими соответственно начальную точку (с парой координат), ширину и высоту , а также методом `MoveTo`, моделирующим перемещение начальной точки в заданную:

```
class Rectangle {
            Point origin;
            public int width, height;
            public Rectangle()
            {origin = new Point(0,0); width=height=0;}
                public Rectangle (Point p, int w, int h)
                    {origin = p; width = w; height = h;}
                public void MoveTo (Point p) {origin = p;}
            }
```

Заметим, что модификатор области видимости для данного класса и его элементов разрешает общедоступное использование (`public`). Рассмотрим пример использования класса `Rectangle`:

```
Rectangle r = new Rectangle(new Point(10,20),5,5);
int area = r.width * r.height;
r.MoveTo(new Point(3, 3));
```

Заметим, что в данном примере последовательно осуществляются инициализация объекта класса `Rectangle` с начальной точкой `(10,20)`, шириной и высотой в пять единиц (т.е. квадрата), подсчет его площади `area` и перемещение начала отсчета в точку с координатами `(3,3)`.

Статические поля объекта в языке C#

Принадлежат классу, а не объекту:

```
class Rectangle {
  static Color defaultColor;// для каждого класса
  static readonly int scale; // для каждого класса
            // статические константы недопустимы
  int x, y, width,height;   // для каждого объекта
    ...
  }
```

Доступ изнутри класса:

```
... defaultColor ... scale ...
```

Доступ из других классов:

```
... Rectangle.defaultColor
... Rectangle.scale ...
```


Комментарий к слайду

В результате анализа рассмотренных примеров становится очевидным, что объект является принципиально динамическим и изменяет состояние в зависимости от соотнесения (времени и внешних воздействий).

В этой связи исследуем более подробно простейший, статический случай полей объекта, который в языке программирования C# выделен в самостоятельный синтаксический элемент, характерный независимостью от состояния объекта (и потому условно принадлежащий к классу). Приведем модифицированный пример предыдущего класса для случая статических полей:

```
class Rectangle {
        static   Color   defaultColor;//   для   каждого   класса
        static readonly int scale; // для каждого класса
        int x, y, width,height;   // для каждого объекта
        ...
}
```

Заметим, что статические поля `defaultColor` и `scale` остаются неизменными внутри класса, тогда время как динамические поля `x`, `y`, `width` и `height` индивидуально изменяются в зависимости от состояния каждого из объектов класса. Доступ изнутри класса осуществляется посредством обращения:

```
        ... defaultColor ... scale ...
```

а из сторонних классов – посредством обращения:

```
        ... Rectangle.defaultColor
        ... Rectangle.scale ...
```

с указанием полных имен объектов. Поскольку статические поля являются неизменными со временем, они реализуются выделением памяти из статической области.

Пример использования метода в языке C#

```
class C {
    int sum = 0, n = 0;
    public void Add (int x)
{   sum = sum + x; n++; // процедура}
    public float Mean(){
     return(float)sum/n; //ф-ция(должна возвратить знач.)
    }
}
```

Доступ изнутри класса:

```
this.Add(3);
float x = Mean();
```

Доступ из других классов:

```
C c = new C();
c.Add(3);
float x = c.Mean();
```


Комментарий к слайду

Познакомившись с особенностями реализации полей как элементов классов с учетом динамики и статики их реализации, перейдем к рассмотрению способов манипулирования объектами классов, которые в объектно-ориентированном программировании принято называть методами, и которые по сути являются функциями. Как и поля, методы описываются в блоке описания класса.

Рассмотрим особенности использования методов на примере следующей программы на языке C#, представляющей описание класса `C` с полями `sum` и `n` и методами `Add` и `Mean`:

```
class C {
     int sum = 0, n = 0;
     public void Add (int x){ sum = sum + x; n++; // процедура}
     public float Mean(){
      return(float)sum/n; //функция(должна возвратить значение)
     }
}
```

Прежде всего, отметим, что методы в языке программирования C# делятся на функции (которые обязаны возвращать значение) и процедуры (которые могут и не возвращать значение, на что указывает тип `void`). В данном примере `Add` – процедура, а `Mean` – функция, в которой возврат значения явно указывается оператором `return`. Доступ к методу, как и к полю, возможно получить изнутри класса:

```
this.Add(3);   float x = Mean();
```

а также из других классов, с явным указанием полного имени:

```
C c = new C(); c.Add(3); float x = c.Mean();
```

Заметим, что оператор `this` представляет собой указатель на текущий объект.

Пример статического метода в языке C#

Операции над данными классов (статическими полями):

```
class Rectangle {
  static Color defaultColor;

  public static void ResetColor() {
        defaultColor = Color.white;
  }
}
```

Доступ изнутри класса:

```
ResetColor();
```

Доступ из других классов:

```
Rectangle.ResetColor();
```


Комментарий к слайду

Продолжая аналогию между полями и методами как элементами классов, мы приходим к понятию статического метода. Рассмотрим особенности реализации статических методов на зыке программирования C# на следующем примере:

```
class Rectangle {
          static Color defaultColor;
          public static void ResetColor() {
               defaultColor = Color.white;
          }
     }
```

Как явствует из приведенного примера, под статическим методом понимается операция, определенная над статическими элементами классов (т.е. над статическими полями). В данном примере описания класса `Rectangle`, статическим является метод `ResetColor` (заметим, что он не возвращает значения, т.е. `ResetColor` – это статическая процедура).

По аналогии с предыдущими случаями, для доступа к статическому методу изнутри класса достаточно указать только краткое имя данного метода:

```
ResetColor();
```

В случае доступа из сторонних классов необходимо указать полное имя статического метода:

```
Rectangle.ResetColor();
```

Обработка и наследование классов и объектов в C#

```
class Stack {
  int[] values;
  int top = 0;
  public Stack(int size) { ... }
  public void Push(int x) {...}
  public int Pop() {...}
}
```

Объекты хранятся в куче (для классов и ссылочных типов)

- Объекты необходимо инициализировать оператором `new`:
  `Stack s = new Stack(100);`
- Классы могут наследовать свойства других классов (единичное наследование кода)
- Классы могут реализовывать множественные интерфейсы (множественное наследование типов)

Комментарий к слайду

Исследовав особенности описания и управление поведением основных элементов классов, объектов и методов, для динамического и статического случаев, кратко остановимся на особенностях наследования свойств (полей и методов) классов объектов языка программирования C#. Для иллюстрации приведем следующий пример фрагмента программы на языке C#:

```
class Stack {
          int[] values;
          int top = 0;
          public Stack(int size) { ... }
          public void Push(int x) {...}
          public int Pop() {...}
     }
```

В данной программе приведено (с сокращениями) описание класса, моделирующего стек (аналогичный стеку КАМ) посредством массива элементов `values` с вершиной `top`, функциями создания стека `Stack` размером `size` и выталкивания `Push` элемента `x` из стека, а также вталкивания `Pop` элемента в стек.

Аналогично объектам ссылочных типов, объекты классов (как принципиально динамические) хранятся в динамической области памяти (куче). В силу ограничений безопасности программного кода любой объект языка программирования C# до использования необходимо инициализировать оператором `new`, например:

```
Stack s = new Stack(100);
```

Заметим, что наследование классами свойств других классов может быть как единичным, так и множественным. Последнее реализуется посредством множественных интерфейсов (приводящим к множественному наследованию типов).

Преимущества и недостатки объектных теорий

Преимущества:

- интуитивная близость произвольной предметной области;
- возможность моделирования сколь угодно сложной предметной области;
- событийно-ориентированный подход;
- высокий уровень абстракции;
- возможность повторного использования описаний;
- параметризация методов обработки объектов

Недостатки:

- сложность тестирования и верификации программ

Комментарий к слайду

Подводя итоги рассмотрения основных понятий объектно-ориентированного подхода к программированию (классов, объектов и методов) применительно к языку программирования C#, кратко отметим достоинства и недостатки подхода.

К преимуществам объектно-ориентированного подхода следует отнести:

- интуитивную близость произвольной предметной области;
- возможность моделирования сколь угодно сложной предметной области, высокий уровень абстракции (рассмотренные примеры дают представление о «масштабируемости» моделирования сложных объектов);
- событийно-ориентированный подход (динамика объектов и возможность манипулирования ими посредством методов приводят к управлению объектами посредством событий);
- возможность повторного использования описаний (основана на обращении к полям и методам извне описания классов, а также на использовании механизма наследования);
- параметризация методов обработки объектов (основана на использовании механизма интерфейсов, которые будут подробно рассмотрены в ходе дальнейших лекций).

К недостаткам объектно-ориентированного подхода к программированию возможно отнести сложность тестирования и верификации программ. Заметим, однако, что выбор ламбда-исчисления и комбинаторной логики в качестве средства формализации объектов, классов и методов позволяет построить адекватную, полную и непротиворечивую объектную модель, учитывающую как статический, так и динамический случаи.

Библиография (1)

1. Schönfinkel M. ‘Über die Bausteine der matematischen Logik, Math. Annalen 92, pp. 305-316, 1924. Translation printed as ‘On the building blocks of mathematical logic’, in van Heijenoort, J. (ed.), From Frege to Gödel, Harvard University Press, 1967
2. Church A. The calculi of lambda-conversion.- Princeton, 1941, ed. 2, 1951
3. Barendregt H.P. The lambda calculus (revised edition), Studies in Logic, 103, North Holland, Amsterdam, 1984
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 таких фундаментальных понятий для каждого языка объектно-ориентированного программирования, как объект, класс и метод. Для более детального ознакомления с особенностями, достижениями и проблемами в теории моделирования этих понятий и практики реализации связанных с ними механизмов рекомендуется следующий список литературы:

1. Schönfinkel M. ‘Über die Bausteine der matematischen Logik, Math. Annalen 92, pp. 305-316, 1924. Translation printed as ‘On the building blocks of mathematical logic’, in van Heijenoort, J. (ed.), From Frege to Gödel, Harvard University Press, 1967
2. Church A. The calculi of lambda-conversion.- Princeton, 1941, ed. 2, 1951
3. Barendregt H.P. The lambda calculus (revised edition), Studies in Logic, 103, North Holland, Amsterdam, 1984
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. Работы [1-3] являются исчерпывающим описанием синтаксиса и семантики ламбда-исчисления, основной формализации языков программирования, в том числе языков ООП. В работе [4] представлен вариант подхода к семантике динамики и статики объектов в форме теории решеток, по которым «протекает» информация.

Библиография (2)

5. Scott D.S. Identity and existence in intuitionistic logic.- In: Application of Sheaves.- Berlin: Springer, 1979, p.p. 600-696
6. Fourman M. The logic of topoi. In: Handbook of Mathematical Logic, J.Barwise et al., eds. North-Holland, 1977
7. Wolfengagen V.E. Event-driven objects. In: Proc. CSIT'1999, Moscow, Russia, 1999, Vol.1, p.p. 88-96
8. Wolfengagen V.E. Fuctional notation for indexed concepts. In: Proc. WFLP'2000, Benicassim, Spain, Sept. 2000. http://www.dsic.upv.es/~wflp2000/
9. Scott D.S. Domains for denotational semantics. ICALP 1982, 577-613

Комментарий к слайду

Продолжим обсуждение работ, посвященных исследованию основных элементов объектно-ориентированного программирования.

5. Scott D.S. Identity and existence in intuitionistic logic.- In: Application of Sheaves.- Berlin: Springer, 1979, p.p. 600-696
6. Fourman M. The logic of topoi. In: Handbook of Mathematical Logic, J.Barwise et al., eds. North-Holland, 1977
7. Wolfengagen V.E. Event-driven objects. In: Proc. CSIT'1999, Moscow, Russia, 1999, Vol.1, p.p. 88-96
8. Wolfengagen V.E. Fuctional notation for indexed concepts. In: Proc. WFLP'2000, Benicassim, Spain, Sept. 2000. http://www.dsic.upv.es/~wflp2000/
9 .Scott D.S. Domains for denotational semantics. ICALP 1982, 577-613

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 определений понятий предметной области, т.е. строится своего рода объектная модель.

В работах [7,8] исследуется событийная ориентированность объектных теорий с учетом динамики и статики объектов, а также их взаимовлияния.

В работе [9] рассматриваются вопросы, связанные с развитием денотационной семантики, имеющей широкое распространение в теории моделирования объектов, и, в частности, с ее представлением посредством доменов.