Объектно-ориентированный подход к программированию

Комментарий к слайду

Предлагаем Вашему вниманию продолжение курса лекций, посвященного современным языкам программирования. Изучение языков программирования базируется на теоретическом фундаменте современных подходов и математических формализаций, принятых в мировом computer science.

В качестве технологической основы и инструментальной платформы для исследования языков программирования предлагается новейшая разработка корпорации Microsoft – уникальный по идеологии и широте поддержки языков программирования комплекс программного обеспечения на основе так называемой методологии .NET.

В первой части курса обсуждались основы программирования и теоретическое введение в computer science.

Вторая часть курса посвящена теории и практике реализации гетерогенных программных систем на более профессиональном уровне. При этом существенно используется объектно-ориентированный подход к программированию, введение в проблематику которого и является темой настоящей лекции.

Содержание лекции

Комментарий к слайду

Коротко о содержании лекции.

Прежде всего, необходимо напомнить слушателям классификацию языков программирования, кратко рассмотреть историю их развития и основные подходы к разработке программных систем, сфокусировав внимание на преимуществах и недостатках каждого из языков и подходов.

При этом особое внимание будет уделено различиям между основными классами языков программирования, а именно, императивными и декларативными.

Затем речь пойдет об отличительных особенностях языков функционального программирования и функциональном подходе в целом.

Основная часть лекции будет посвящена концепциям объектно-ориентированного программирования, а именно, наследования, инкапсуляции и полиморфизма. По традиции, примеры программ (на языке программирования C#, самом популярном для платформы .NET) будут сопровождаться теоретическими аналогиями из computer science. Исследование эволюции языков программирования данного класса будет сопровождаться примерами.

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

Подходы к программированию

Основные подходы к программированию:

- структурный, модульный;
- функциональный;
- логический;
- объектно-ориентированный (ООП);
- смешанный (комбинированный, интегрированный);
- компонентно-ориентированный (.NET);
- чисто объектный

Комментарий к слайду

Прежде всего, напомним слушателям итоги построения классификации языков и подходов к программированию, разработанной нами в первой части курса.

Кратко перечислим основные подходы к программированию:

- ранние неструктурные подходы;
- структурный или модульный подход (задача разбивается на подзадачи, затем на алгоритмы, составляются их структурные схемы и происходит реализация);
- функциональный подход;
- логический подход;
- объектно-ориентированный подход;
- смешанный подход (некоторые подходы возможно комбинировать);
- компонентно-ориентированный (программный проект рассматривается как множество компонент, такой подход принят, в частности, в .NET);
- чисто объектный подход (идеальный с математической точки зрения вариант, который пока не реализован практически).

Заметим, что приведенную классификацию не следует считать единственно верной и абсолютной, поскольку языки программирования постоянно развиваются и совершенствуются, и недавние недостатки устраняются с появлением необходимых инструментальных средств или теоретических обоснований.

Декларативные языки программирования (1)

Время появления: 1960-е г.г.

Краткая характеристика: программа – описание действий, которые необходимо осуществить

Преимущества:

- простота верификации и тестирования программ;
- строгость математической формализации;
- высокая степень абстракции

Комментарий к слайду

Остановимся несколько подробнее на языках и подходах к программированию, которые наиболее существенны для целей данного курса.

Напомним, что в 60-х г.г. возникает новый подход к программированию, который до сих пор успешно конкурирует с императивным (т.е. основанным на командах, директивах или операторах и включающим процедуры и функции), а именно, декларативный подход.

Суть подхода состоит в том, что программа представляет собой не набор команд, а описание действий, которые необходимо осуществить.

Этот подход, как показали результаты предыдущего курса, существенно проще и прозрачнее формализуем математическими средствами. Отсюда следует тот факт, что программы проще проверять на наличие ошибок (тестировать), а также на соответствие заданной технической спецификации (верифицировать).

Высокая степень абстракции также является преимуществом данного подхода. Фактически, программист оперирует не набором инструкций, а абстрактными понятиями, которые могут быть достаточно обобщенными.

Декларативные языки программирования (2)

Недостатки:

- сложность эффективной реализации;
- необходимость фундаментальных математических знаний

Примеры:

LISP (Interlisp, Common Lisp, Scheme), SML, Haskell, Prolog

Комментарий к слайду

На начальном этапе развития декларативным языкам программирования было сложно конкурировать с императивными в силу объективных трудностей при создании эффективной реализации трансляторов. Программы работали медленнее, однако, они могли решать с меньшими трудозатратами более абстрактные задачи.

В частности, язык SML, который мы продолжаем изучать в данном курсе вместе с языком C#, был разработан как средство доказательства теорем.

Различные диалекты языка LISP (основные из них представлены на слайде), возникли потому, что ядро и идеология этого языка оказались весьма эффективными при реализации символьной обработки (анализе текстов).

Другие характерные примеры декларативных языков программирования приведены на слайде.

Императивные (процедурные) языки программирования (1)

Время появления: 1950-е г.г.

Краткая характеристика:

программа – последовательность инструкций-***операторов***, включающих блоки типичных действий – процедуры или функции

Преимущества:

- более высокий уровень абстракции;
- меньшая машинная зависимость;
- более широкая совместимость

Комментарий к слайду

50-е годы XX века ознаменовались появлением языков программирования так называемого «высокого уровня», по сравнению с ранее рассмотренными нами низкоуровневыми языками.

При этом различие состоит в повышении эффективности труда разработчиков за счет абстрагирования или отвлечения от конкретных деталей аппаратного обеспечения. Одна инструкция (оператор) языка высокого уровня соответствовала последовательности из нескольких низкоуровневых инструкций, или команд. Исходя из того, что программа, по сути, представляла собой набор директив, обращенных к компьютеру, такой подход к программированию получил название императивного.

Еще одной особенностью языков высокого уровня была возможность повторного использования ранее написанных программных блоков, выполняющих те или иные действия, посредством их идентификации и последующего обращения к ним, например, по имени. Такие блоки получили название функций или процедур, и программирование приобрело более упорядоченный характер.

Императивные (процедурные) языки программирования (2)

Преимущества:

- содержательная значимость текстов программ;
- унификация программного кода;
- повышение производительности труда программистов

Недостатки:

- большие трудозатраты на обучение;
- меньшая эффективность программного кода

Примеры:

Fortran, ALGOL, PL/1, APL, BPL, COBOL, Pascal, C, Basic

Комментарий к слайду

Кроме того, с появлением языков высокого уровня зависимость реализации от аппаратного обеспечения существенно уменьшилась. Платой за это стало появление специализированных программ, преобразующих инструкции возникших языков в коды той или иной машины, или трансляторов, а также некоторая потеря в скорости вычислений, которая, впрочем, компенсировалась существенным выигрышем в скорости разработки приложений и унификацией программного кода.

Нужно отметить, что операторы и ключевые слова новых языков программирования были более осмысленны, чем безликие цифровые последовательности машинных кодов, что также обеспечивало повышение производительности труда программистов.

Естественно, для обучения новым языкам программирования требовались значительные затраты времени и средств, а эффективность реализации на прежних аппаратных возможностях снижалась. Однако трудности эти носили временный характер, и, как показала практика программирования, многие из первых языков высокого уровня оказались настолько удачно реализованными, что активно используются и сегодня.

Одним из таких примеров является язык Fortran, реализующий вычислительные алгоритмы. Другой пример - язык APL, трансформировавшийся в BPL и затем в C. Основные конструкции последнего остаются неизменными вот уже несколько десятилетий и присутствуют в языке C#, который нам предстоит изучать в курсе объектно-ориентированного программирования.

Примеры других языков программирования: ALGOL, COBOL, Pascal, Basic.

Функциональные языки программирования (1)

Время появления: 1960-е г.г.

Краткая характеристика:

программа – функция, аргументы которой, возможно, также являются функциями

Преимущества:

- полностью автоматическое управление памятью компьютера («сборка мусора»);
- простота повторного использования фрагментов кода;
- расширенная поддержка функций с параметрическими аргументами (параметрический полиморфизм);

Комментарий к слайду

Одним из путей развития декларативного стиля программирования стал функциональный подход, возникший с появлением и развитием языка LISP.

Отличительной особенностью данного подхода является то обстоятельство, что любая программа, написанная на таком языке, может интерпретироваться как функция с одним или несколькими аргументами. Такой подход дает возможность прозрачного моделирования текста программ математическими средствами, а, значит, весьма интересен с теоретической точки зрения.

Сложные программы при таком подходе строятся посредством агрегирования функций. При этом текст программы представляет собой функцию, некоторые аргументы которой можно также рассматривать как функции. Таким образом, повторное использование кода сводится к вызову ранее описанной функции, структура которой, в отличие от процедуры императивного языка, прозрачна математически.

Более того, типы отдельных функций, используемых в функциональных языках, могут быть переменными. Таким образом обеспечивается возможность обработки разнородных данных (например, упорядочение элементов списка по возрастанию для целых чисел, отдельных символов и строк) или полиморфизм.

Еще одним важным преимуществом реализации языков функционального программирования является автоматизированное динамическое распределение памяти компьютера для хранения данных. При этом программист избавляется от рутинной обязанности контролировать данные, а при необходимости может запустить функцию «сборки мусора» – очистки памяти от тех данных, которые больше не потребуются программе (обычно этот процесс периодически инициируется компьютером).

Функциональные языки программирования (2)

Преимущества:

- абстрагирование от машинного представления данных;
- прозрачность реализации самоприменяемых (рекурсивных) функций;
- удобство символьной обработки данных (списки, деревья)

Недостатки:

- нелинейная структура программы;
- относительно низкая эффективность

Примеры:

LISP, SML, CaML, Haskell, Miranda, Hope

Комментарий к слайду

Таким образом, при создании программ на функциональных языках программист сосредотачивается на области исследований (предметной области) и в меньшей степени заботится о рутинных операциях (обеспечении правильного с точки зрения компьютера представления данных, «сборке мусора» и т.д.).

Поскольку функция является естественным формализмом для языков функционального программирования, реализация различных аспектов программирования, связанных с функциями, существенно упрощается. В частности, интуитивно прозрачным становится написание рекурсивных функций, т.е. функций, вызывающих самих себя в качестве аргумента. Кроме того, естественной становится и реализация обработки рекурсивных структур данных (например, списков – базовых элементов, скажем, для семейства языков LISP, деревьев и др.)

Благодаря реализации механизма сопоставления с образцом, такие языки как ML и Haskell весьма хорошо применимы для символьной обработки.

Естественно, языки функционального программирования не лишены недостатков. Часто к ним относят нелинейную структуру программы и относительно невысокую эффективность реализации. Однако, первый недостаток достаточно субъективен, а второй успешно преодолен современными реализациями, в частности, рядом последних трансляторов языка SML, включая и компилятор для среды Microsoft .NET.

Характерные примеры функциональных языков программирования приведены на слайде.

Объектно-ориентированные языки программирования (1)

Время появления: 1970-е г.г.

Краткая характеристика:

программа – описание объектов, их совокупностей, отношений между ними и способов их взаимодействия

Преимущества:

- интуитивная близость к произвольной предметной области;
- моделирование сколь угодно сложных предметных областей;
- событийная ориентированность;

Комментарий к слайду

Важнейшим шагом на пути к совершенствованию языков программирования стало появление объектно-ориентированного подхода к программированию (или, сокращенно, ООП) и соответствующего класса языков. Именно исследование теории и практики проектирования и реализации программных систем по принципам ООП и является основной целью данного курса.

При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

Объектно-ориентированные языки программирования (2)

Преимущества:

- высокий уровень абстракции;
- повторное использование описаний;
- параметризация методов обработки объектов

Недостатки:

- сложность тестирования и верификации программ

Примеры:

C++, Visual Basic, C#, Eiffel, Oberon

Комментарий к слайду

Перемещаясь по иерархии классов от более общих понятий предметной области к более конкретным (или от более сложных – к более простым) и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый им реальный мир.

Использование ранее разработанных (возможно, другими коллективами программистов) библиотек объектов и методов позволяет значительно сэкономить трудозатраты при производстве программного обеспечения, в особенности, типичного.

Объекты, классы и методы могут быть полиморфными, что делает реализованное программное обеспечение более гибким и универсальным.

Сложность адекватной (непротиворечивой и полной) формализации объектной теории порождает трудности тестирования и верификации созданного программного обеспечения. Пожалуй, это обстоятельство является одним из самых существенных недостатков объектно-ориентированного подхода к программированию.

Наиболее известным примером объектно-ориентированного языка программирования является язык C++, развившийся из императивного языка C. Его прямым потомком и логическим продолжением является язык C#, который исследуется в качестве образца в данном курсе. Другие примеры объектно-ориентированных языков программирования представлены на слайде.

Принципы объектно-ориентированного программирования:

1. Абстракция данных

2. Наследование конкретных атрибутов объектов и функций оперирования объектами на основе иерархии

3. Инкапсуляция (свойства и методы «спрятаны» внутри объекта)

4. Полиморфизм (функции с возможностью обработки данных переменного типа)

Комментарий к слайду

Переход от структурно-процедурного подхода к программированию к объектно-ориентированному, подобно переходу от низкоуровневых языков программирования к языкам высокого уровня, требует значительных затрат времени и сил на обучение. Естественно, что платой за полученные знания является повышение производительности труда программистов при проектировании и реализации программного обеспечения. Другим преимуществом, которое дает ООП перед императивным подходом, является относительно более высокий процент повторного использования уже разработанного программного кода.

При этом, в отличие от предыдущих подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или, иначе, концепций, на которых он зиждется. К числу основополагающих понятий ООП обычно относят абстракцию данных, наследование, инкапсуляцию и полиморфизм.

Зачастую в практических и учебных курсах по программированию слушатели не имеют четкого математического основания для формирования достаточно полного и ясного представления об основах ООП. Преимуществом предлагаемого курса является то обстоятельство, что уже изученные в первой части курса разделы computer science (такие как, например, ламбда-исчисление и комбинаторная логика) позволяют сформировать глубокое и точное понимание фундаментальных понятий объектно-ориентированного программирования. В частности, понятие абстракции – основной операции ламбда-исчисления – для нас является уже хорошо знакомым.

Поясним на качественном уровне другие фундаментальные принципы ООП. Так, наследование конкретных атрибутов объектов и функций оперирования объектами основано на иерархии. Инкапсуляция означает «сокрытие» свойств и методов внутри объекта. Полиморфизм, как и в функциональном программировании, понимается как наличие функций с возможностью обработки данных переменного типа.

Абстракция и методы ее моделирования

Вообще говоря, под ***абстракцией*** понимается выражение языка программирования, отличное от идентификатора.

Значение функции или переменной может быть присвоено абстракции и является значением последней.

Поведение абстракции заключается в приложении функции к аргументу.

Абстракция адекватно моделируется ламбда-исчислением (а именно, посредством операции абстракции).

Комментарий к слайду

Рассмотрим более подробно такой фундаментальный принцип объектно-ориентированного подхода к программированию, как абстракция.

В разделах математики, исследующих моделирование процесса создания программ, под абстракцией принято понимать произвольное выражение языка программирования, которое является отличным от идентификатора.

Важнейшей операцией, которая была исследована нами в первой части курса, является операция вычисления значения выражения или команды, т.е. операция означивания (в частности, функция вычисления значения явно используется при построении семантики языка программирования). В этой связи важно установить, что является значением абстракции. Будем считать, что значение функции или переменной может быть присвоено абстракции и является значением последней.

В объектно-ориентированном программировании каждый объект является принципиально динамической сущностью, т.е. изменяется в зависимости от времени (а также от воздействия внешних по отношению к нему факторов). Иначе говоря, объект обладает тем или иным образом поведения. В отношении абстракции как объекта, поведение заключается в приложении функции к аргументу.

Как мы уже отмечали, концепция абстракции в объектно-ориентированном программировании адекватно моделируется посредством ламбда-исчисления. Точнее говоря, операция абстракции в полной мере является моделью одноименного понятия ООП.

Наследование и методы его моделирования

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

В языках программирования понятие наследование означает применимость (некоторых) свойств или методов базового класса для классов, производных от него (а также для их конкретизаций).

Наследование моделируется (иерархическим) отношением частичного порядка и адекватно формализуется посредством:

1) фреймовой нотации Руссопулоса (N.D. Roussopulos);
2) диаграмм Хассе (Hasse)

Комментарий к слайду

Другой фундаментальной составляющей концепции объектно-ориентированного программирования является интуитивно ясное понятие наследования.

В неформальной постановке под ***наследованием*** понимается свойство того или иного объекта, который является производным от некоего базового, сохранять поведение (а именно, атрибуты и операции над ними), характерное для родительского объекта.

С точки зрения языков программирования понятие наследования означает применимость всех или лишь некоторых свойств и/или методов базового (или родительского) класса для всех классов, производных от него. Кроме того, сохранение свойств и/или методов базового класса должно обеспечиваться и для всех конкретизаций (т.е. конкретных объектов) любого производного класса.

В математике концепцию наследования принято моделировать, например, отношением частичного порядка (которое представляет собой вид иерархии). Концепцию наследования можно адекватно формализуется математически посредством одной из следующих нотаций:

1) фреймовой нотации Руссопулоса (названной так по имени своего создателя, N.D. Roussopulos);

2) диаграмм Хассе (получивших название по имени ученого, который впервые предложил этот способ наглядного представления наследования Helmut Hasse).

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

Рассмотрим пример программы на языке программирования C#, иллюстрирующий концепцию наследования:

```
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() {...}
     }
```

Пример представляет собой описание базового класса `A` и производного от него класса `B`.

Класс `A` содержит целочисленный атрибут (т.е. переменную) `a`, а также два метода (т.е. функции) `A(…)` и `F(…)`. Класс `B` содержит целочисленный атрибут (т.е. переменную) `b`, а также два метода (т.е. функции) `B(…)` и `G(…)`.

Двоеточие `B:A` в описании класса `B` означает наследование.

Отметим, что в заголовок слайда вынесен простейший случай наследования, а именно, единичное наследование. Язык программирования C# позволяет реализовать механизмы, поддерживающие и более сложный случай наследования, а именно, множественное наследование.

Пример единичного наследования на C# (2)

B 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

Рассмотрим более подробно, особенности наследования, которые реализует данный пример программы на языке C#.

Производный класс B наследует от базового класса A свойство a и метод F(). При этом к классу B добавляются собственные свойство b и метод G().

Заметим, что в отношении операции наследования справедливы следующие ограничения:
1) конструкторы (т.е. функции создания и инициализации классов) не наследуются;
2) в языке C# существует возможность замещения наследуемых методов (этот языковой аспект будет рассмотрен более подробно в ходе дальнейших лекций).

Отметим также, что наиболее простым случаем наследования является так называемое единичное наследование. При таком наследовании производный класс (или, иначе, подкласс) может наследовать свойства только одного базового класса. Однако при этом производный класс может этом реализовывать множественные интерфейсы (т.е. использовать описания объектов и методов других классов, напрямую минуя механизм наследования).

Один класс языка программирования C# может наследовать лишь свойства другого класса (но не структуры – типа данных, аналогичного кортежу языка программирования SML).

Структура не может наследовать свойства другого типа данных, однако может при этом реализовывать как один, так и несколько интерфейсов.

Подкласс с неявным базовым классом наследует свойства наиболее абстрактного класса, известного под названием «объект» (`object`).

Понятие инкапсуляции в программировании

Вообще говоря, под ***инкапсуляцией*** понимается доступность объекта исключительно посредством его свойств и методов.

Таким образом, свойствами объекта (явно описанными или производными) возможно оперировать исключительно посредством его методов.

Свойства инкапсуляции:

- совместное хранение данных и функций;
- сокрытие внутренней информации от пользователя;
- изоляция пользователя от особенностей реализации

Комментарий к слайду

Еще одним фундаментальным компонентом концепции объектно-ориентированного программирования является понятие инкапсуляции.

Неформально говоря, под инкапсуляцией понимается возможность доступа к объекту и манипулирования им исключительно посредством предоставляемых именно этим объектом свойств и методов.

Таким образом, свойствами объекта (безразлично, являются ли они явно описанными или производными от других объектов) возможно оперировать исключительно посредством методов, которые содержатся в описании этого объекта (или родительских по отношению к нему объектов, при условии, что эти методы унаследованы).

Инкапсуляция является весьма важным свойством, поскольку обеспечивает определенную (а точнее, определяемую программистом) степень безопасности данных об объекте.

Хотя инкапсуляция как таковая является фундаментальным свойством ООП, степень инкапсуляции при наследовании может варьироваться в зависимости от типа области видимости объекта, который определяется модификатором видимости. Так, используемый в предыдущем примере модификатором видимости `public` обеспечивает доступность свойств и методов объекта из произвольного места программы.

К основным свойствам инкапсуляции относятся следующие возможности:

1) совместное хранение данных и функций (т.е. свойств и методов внутри объекта);
2) сокрытие внутренней информации от пользователя (что обеспечивает большую безопасность приложения);
3) изоляция пользователя от особенностей реализации (что обеспечивает машинную независимость и потенциально дружественный интерфейс приложений).

Понятие полиморфизма в программировании

Вообще говоря, под ***полиморфизмом*** понимается возможность оперировать объектами, не обладая точным знанием их типов.

Рассмотрим пример полиморфной функции:

```
void Poly(object o) {
  Console.WriteLine(o.ToString());
}
```

а также вариантов ее использования:

```
Poly(25);
Poly("John Smith");
Poly(3.141592536m);
Poly(new Point(12,45));
```


Комментарий к слайду

Как нам уже известно, важной позитивной особенностью языка программирования SML является то обстоятельство, что в нем поддерживается так называемая полиморфная типизация, суть которой можно объяснить на основе следующего примера. Рассмотрим функцию обработки списка, которая упорядочивает его элементы по возрастанию. В классическом языке программирования со строгой типизацией неизбежно придется реализовать по крайней мере две функции: для случаев числовых и строковых элементов списка. В SML не возникает такой необходимости, т.к. существует возможность описания функции обработки списка с переменным типом аргументов, которая безошибочно обработает и список из чисел, и список из строк.

В объектно-ориентированном программировании под полиморфизмом понимается возможность оперировать объектами, не обладая точным знанием их типов. Рассмотрим пример простейшей полиморфной функции:

```
void Poly(object o) {
  Console.WriteLine(o.ToString());
}
```

Данная функция реализует выдачу на экран объекта (метод `Console.WriteLine`) с предварительным преобразованием его к строковому типу (метод `ToString()`).

Все приведенные на слайде варианты вызова функции:

```
Poly(25);
Poly("John Smith");
Poly(3.141592536m);
Poly(new Point(12,45));
```

корректно пройдут компиляцию и завершатся выдачей корректного результата.

Библиография (1)

1. Pratt T.W., Zelkovitz M.V. Programming languages, design and implementation (4th ed.).- Prentice Hall, 2000

2. Appleby D., VandeKopple J.J. Programming languages, paradigm and practice (2nd ed.).- McGraw-Hill, 1997

3. Troelsen A. C# and the .NET platform (2nd ed.).- APress, 2003, 1200 p.p.

4. Liberty J. Programming C# (2nd ed.).- O'Reilly, 2002, 656 p.p.

Комментарий к слайду

К сожалению, в рамках одной лекции невозможно в полном объеме представить глубину объектно-ориентированного подхода к программированию. Мы ограничились кратким рассмотрением ООП в системе возможных подходов, а также формированием представления об основополагающих принципах объектно-ориентированного подхода.

Для более детального ознакомления с последними достижениями и проблемами в области ООП, рекомендуется следующий список литературы:

1. Pratt T.W., Zelkovitz M.V. Programming languages, design and implementation (4th ed.).- Prentice Hall, 2000

2. Appleby D., VandeKopple J.J. Programming languages, paradigm and practice (2nd ed.).- McGraw-Hill, 1997

3.Troelsen A. C# and the .NET platform (2nd ed.).- APress, 2003, 1200 p.p.

4. Liberty J. Programming C# (2nd ed.).- O'Reilly, 2002, 656 p.p.

Кратко остановимся на источниках. В работе [1] приведен наиболее полный анализ истории развития и особенностей языков программирования с классификацией по областям применения. В работах [2-4] рассмотрены теоретические проблемы и практические аспекты реализации инновационных конструкций в языках программирования, прежде всего, в языке C#.NET, изучение основ которого планируется в данном семестре.

Библиография (2)

5. Landin P. The next 700 programming languages. Communications of ACM, 3, 1966

6. Peyton Jones S.L. The implementation of functional programming languages.- Prentice Hall, 1987

7. Gilmore S. Programming in Standard ML ’97: a tutorial introduction. http:/www.dcs.ed.ac.uk/home/stg

Комментарий к слайду

Продолжим обсуждение библиографии.

5.Landin P. The next 700 programming languages. Communications of ACM, 3, 1966

6.Peyton Jones S.L. The implementation of functional programming languages.- Prentice Hall, 1987

7.Gilmore S. Programming in Standard ML ’97: a tutorial introduction. http:/www.dcs.ed.ac.uk/home/stg

В работе [5] исследуются современные тенденции развития языков и подходов к программированию.

Работа [6] посвящена вопросам реализации языков программирования, в частности, на основе функционального подхода.

Работа [7] содержит описание языка функционального программирования Standard ML (в сравнении с которым в данном курсе излагается ООП) и практике программирования на нем.