Концепция инкапсуляции 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 инкапсуляции – одной из фундаментальных концепций, на которых базируется 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, инкапсуляции.

Прежде всего, будет сформулировано определение понятия инкапсуляции.

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

Особое внимание будет уделено реализации механизмов инкапсуляции для объектов языка программирования C#, включая классификацию областей видимости объектов, а также особенности их применения.

Рекомендации по использованию областей видимости с примерами программ на языке C# проиллюстрируют практику применения концепции инкапсуляции.

Лекция завершится обзором литературы для более глубокого исследования материала.

Концепция 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

Прежде всего, попытаемся формализовать понятие инкапсуляции в рамках объектно-ориентированного подхода к программированию.

В неформальной постановке вопроса будем понимать под инкапсуляцией доступность объекта исключительно посредством его свойств и методов.

Другими словами, концепция инкапсуляции призвана обеспечивать безопасность проектирования и реализации программного обеспечения на основе локализации манипулирования объектом в областях его полей и методов.

Иначе говоря, свойствами объекта возможно оперировать исключительно посредством его методов. Это замечание касается как свойств, явно определенных в описании объекта, так и свойств, унаследованных данным объектом от другого (других).

Практическая важность концепции инкапсуляции для современных языков объектно-ориентированного программирования (в том числе и для языка C#) определяется следующими фундаментальными свойствами.

Прежде всего, реализация концепции инкапсуляции обеспечивает совместное хранение данных (или, иначе, полей) и функций (или, иначе, методов) внутри объекта.

Как следствие, механизм инкапсуляции приводит к сокрытию информации о внутреннем «устройстве» объекта данных (или, в терминах языков ООП, свойств и методов объекта) от пользователя того или иного объектно-ориентированного приложения.

Таким образом, пользователь, получающий программное обеспечение как сервис, оказывается изолированным от особенностей среды реализации.

Понятие инкапсуляции в computer science

Формальные модели инкапсуляции:

1. Ламбда-исчисление:

- ламбда-термы выполняют роль объектов;
- связанные переменные выполняют роль свойств;
- свободные переменные выполняют роль методов;

2. Комбинаторная логика:

- комбинаторы выполняют роль объектов;
- переменные выполняют роль свойств;
- комбинаторы выполняют роль методов.

Комментарий к слайду

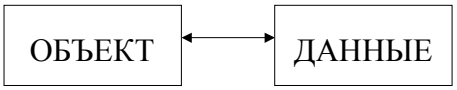
Обсудив определение понятия инкапсуляции (пока на интуитивном уровне), перейдем к рассмотрению формализаций этой фундаментальной для объектно-ориентированного программирования концепции на основе уже известных нам формальных теорий computer science.

Как оказывается, понятие инкапсуляции вполне адекватно формализуется посредством таких теоретических формальных систем, как ламбда-исчисление А.Черча и комбинаторная логика Х.Карри.

При этом, при интерпретации концепции инкапсуляции в терминах формальной системы ламбда-исчисления, в роли объектов выступают ламбда-термы, в роли свойств – связанные переменные, а в роли методов – свободные переменные.

В случае же интерпретации концепции инкапсуляции в терминах формальной системы комбинаторной логики, в роли объектов выступают комбинаторы, в роли свойств – переменные, а в роли методов – комбинаторы.

Объект традиционного языка (сокрытие информации)



В традиционных языках программирования объявления данных и процедуры обработки этих данных отделены друг от друга.

Доступ к данным может быть получен «со стороны», вообще говоря, посредством произвольной процедуры, что является непоследовательным и небезопасным.

Комментарий к слайду

Поясним в более детальной форме реализацию механизма сокрытия информации посредством концепции инкапсуляции.

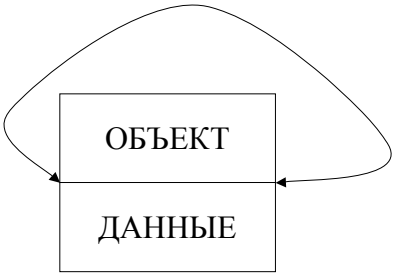
Рассмотрим в достаточно обобщенном виде схему взаимодействия объекта и данных.

Вначале представим схему такого рода для традиционного императивного языка программирования (каковыми являются, например, языки C и Pascal). При этом в нашем рассуждении под термином «объект» будем понимать произвольный объект языка программирования, безотносительно к концепции объектно-ориентированного программирования.

В этом случае, объявления данных и процедуры обработки данных отделены друг от друга. Зачастую в ранних языках программирования описания языковых объектов и процедуры манипулирования ими выделены в обособленные разделы.

В этой связи, принципиальным недостатком ранних императивных языков программирования является то обстоятельство, что доступ к данным может быть получен методами, которые изначально не предназначались разработчиками приложений для манипулирования этими данными. Вообще говоря, управление объектом может осуществляться посредством произвольной процедуры или функции, и у среды проектирования и реализации программного обеспечения нет возможности централизованного управления этим процессом. Такой подход к программированию, безусловно, является довольно непоследовательным и весьма небезопасным.

Схема инкапсуляции объекта языка ООП



В ООП определение и методы обработки свойств объекта хранятся совместно; доступ к объекту, минуя его методы, не представляется возможным.

Комментарий к слайду

В отличие от предыдущей схемы, при объектно-ориентированном подходе к проектированию и реализации прикладного программного обеспечения взаимодействие объектов языка программирования и конкретизирующих его данных реализуется принципиально иным образом.

По существу, объект и данные составляют единое и неделимое целое.

Прежде всего, определение (или описание свойств классов) и процедуры манипулирования этими свойствами (или методы) для каждого объекта языка программирования при объектно-ориентированном подходе хранятся совместно.

Кроме того, среда проектирования и реализации программного обеспечения (например, Microsoft Visual Studio .NET) не предоставляет иных возможностей доступа к объекту, как посредством методов, изначально предназначенных для манипулирования данным объектом.

Таким образом, концепция инкапсуляции в языках объектно-ориентированного программирования служит целям обеспечения единообразия определения, хранения и манипулирования информацией о языковых объектах.

Основные виды областей видимости в языке C#

Степень инкапсуляции объекта зависит от вида области видимости:

1. `public` (доступность из любого места, для которого известно пространство имен с описанием объекта):

- элементы интерфейсов и перечислений являются `public` по умолчанию;

- типы в пространствах имен (классы, структуры, интерфейсы, перечисления, делегаты) по умолчанию являются доступными как `internal` (видимы из сборки с описанием объекта);

2. `private` (доступность из описания класса или структуры):

- элементы классов и структур (поля, методы, свойства, вложенные типы и т.д.) являются `private` по умолчанию.

Комментарий к слайду

Заметим, что инкапсуляция является безусловно необходимым требованием для каждого объекта.

В то же время, в практике программирования, степень инкапсуляции объекта (в широком значении этого слова) определяется его описанием, а также описанием порождающих его объектов (в действительности это, как правило, описания базовых и производных классов).

В этой связи в языках объектно-ориентированного программирования вводится понятие области видимости как степени доступности произвольного языкового объекта.

Применительно к языку программирования C# области видимости объектов подразделяются на следующие виды.

Общедоступные объекты описываются с помощью зарезервированного слова `public` и характеризуются доступностью из произвольного места программы, для которого определено пространство имен с описанием рассматриваемого объекта. При этом элементы интерфейсов и перечислений являются общедоступными (`public`) объектами по умолчанию.

С другой стороны, типы, описанные в составе пространств имен в форме классов, структур, интерфейсов, перечислений или делегатов являются по умолчанию видимыми из сборки с описанием объекта и описываются с помощью зарезервированного слова `internal`.

Наконец, элементы классов и структур, в частности, поля, методы, свойства и вложенные типы являются по умолчанию доступными из описаний соответствующих классов или структур и описываются с помощью зарезервированного слова `private`.

Пример использования областей видимости в C#

```
public class Stack {
 private int[] val;
    // private используется и по умолчанию
 private int top;
    // private используется и по умолчанию
 public Stack() {...}
 public void Push(int x) {...}
 public int Pop() {...}
}
```


Комментарий к слайду

Проиллюстрируем обсуждение использования модификаторов областей видимости объектов (`public` и `private`) следующим содержательным примером фрагмента программы на языке C#:

```
public class Stack
      {
        private int[] val;
           // private используется и по умолчанию
        private int top;
           // private используется и по умолчанию
        public Stack() {...}
        public void Push(int x) {...}
        public int Pop() {...}
      }
```

Как видно из приведенного примера, фрагмент программы на языке C# содержит описание класса стека `Stack`, реализованного на основе массива целочисленных элементов (поле `val`). Голова стека представляет собой целочисленное значение (поле `top`). Над стеком определены операции инициализации (метод `Stack`), а также вставки (метод `Push`) и удаления (метод `Pop`) элемента.

Как явствует из примера, класс `Stack` и все манипулирующие его объектами методы (`Stack`, `Push` и `Pop`) являются общедоступными (`public`), тогда как поля являются доступными локально (`private`), т.е. только из описания данного класса.

Расширенные возможности использования областей видимости в языке C#

Виды дополнительных областей видимости объектов в языке C#:

1) `protected`: доступность из класса с описанием объекта и его подклассов;
2) `internal`: доступность из сборки с описанием объекта;
3) `protected internal`: доступность из класса с описанием объекта, его подклассов, а также из сборки с описанием объекта

Комментарий к слайду

Заметим, что в приведенном выше примере фрагмента программы с описанием областей видимости использовались только базовые возможности модификаторов видимости языка программирования C#.

Оказывается, что язык программирования C# располагает механизмами реализации дополнительных (или расширенных) по сравнению с базовыми областей видимости объектов.

Рассмотрим более подробно особенности основных типов расширенных областей видимости объектов в языке программирования C#.

К числу расширенных областей видимости следует отнести доступность языковых объектов как непосредственно из класса с описанием объекта, так и из его производных классов. В данном случае для описания языкового объекта используется зарезервированное слово `protected`.

Кроме того, для описания доступности языкового объекта лишь из сборки с его описанием используется зарезервированное слово `internal`.

Наконец, для описания доступности языкового объекта непосредственно из класса с описанием данного объекта, его производных классов, а также из сборки с описанием данного объекта используется зарезервированное слово `protected internal`.

Пример использования расширенных областей видимости в языке C#

```
class Stack {
  protected int[] values = new int[32];
  protected int top = -1;
  public void Push(int x) {...}
  public int Pop() {...}
}

class BetterStack : Stack {
   public bool Contains(int x) {
   foreach (int y in values) if(x==y) return true;
   return false;
  }
}

class Client {
  Stack s = new Stack();
  ... s.values[0] ... // ошибка при компиляции!
}
```


Комментарий к слайду

Проиллюстрируем обсуждение использования модификаторов расширенных областей видимости объектов (`protected` и `internal`) следующим содержательным примером фрагмента программы на языке C#:

```
class Stack {
        protected int[] values = new int[32];
        protected int top = -1;
        public void Push(int x) {...}
        public int Pop() {...}
      }
      class BetterStack : Stack {
           public bool Contains(int x) {
           foreach (int y in values) if(x==y) return true;
           return false;
        }
      }
      class Client {
        Stack s = new Stack();
        ... s.values[0] ...  // ошибка при компиляции!
      }
```

Как видно из приведенного примера, фрагмент программы на языке C# содержит описание класса стека `Stack` (для хранения `32` целочисленных элементов и значением `-1` в вершине), а также реализованного на его основе усовершенствованного класса стека `BetterStack`, дополнительно реализующего повторяющиеся элементы стека. В отличие от предыдущего примера, все поля класса стека `Stack` доступны как из данного класса, так и из классов, производных от него, поскольку описаны посредством ключевого слова `protected`. Особенности реализации оператора `foreach` будут рассмотрены далее в ходе курса.

Инкапсуляция полей и констант в языке C# (1)

Поле:

- инициализация факультативна, однако запрещен доступ к полям и методам того же типа;
- поля структуры не подлежат инициализации

Пример поля:

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

Константа:

- Значение должно быть вычислимо в процессе компиляции

Пример константы:

```
const long size = ((long)int.MaxValue + 1) / 4;
```


Комментарий к слайду

После обсуждения использования (степеней) инкапсуляции применительно к классам в целом, рассмотрим особенности приложения данной концепции к таким фундаментальным объектам языка программирования C# как поля и константы.

Инициализация не является безусловно необходимой для полей в языке программирования C#. Тем не менее, доступ к полям и методам изначально запрещен (что соответствует использованию по умолчанию модификатора доступа `private`). В случае структуры поля инициализации не подлежат.

Простейшей иллюстрацией описания поля в языке программирования C# является следующий пример, содержащий определение класса `C` с целочисленным полем `value`:

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

В случае константы инициализация также не является необходимым требованием. Тем не менее, значение константы должно быть вычислимо в процессе компиляции.

Простейшей иллюстрацией описания константы в языке программирования C# является следующий пример, содержащий определение константы `size` , представляющей собой целочисленное значение двойной длины (`long`):

```
const long size = ((long)int.MaxValue + 1) / 4;
```

Заметим, что фрагмент …`(long)` … в правой части присваивания представляет собой явное преобразование типов языковых объектов.

Инкапсуляция полей и констант в языке C# (2)

Поле только для чтения (`readonly`):

- необходимо инициализировать в описании или конструкторе;
- значение не обязательно должно быть вычислимым в ходе компиляции;
- занимает область памяти (аналогично полю).

Пример поля только для чтения: `readonly DateTime date;`

Доступ изнутри класса: `... value ... size ... date ...`

Доступ из других классов:

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


Комментарий к слайду

Особым случаем реализации концепции инкапсуляции в языке объектно-ориентированного программирования C# является механизм полей, предназначенных только для чтения. Для описания такого рода полей в языке C# используется зарезервированное слово `readonly`.

Основные свойства полей, предназначенных только для чтения, сводятся к следующему. Прежде всего, такие поля необходимо инициализировать непосредственно при описании или в конструкторе класса. Кроме того, значение поля, предназначенного только для чтения, не обязательно должно быть вычислимым в ходе компиляции, а может быть вычислено во время выполнения программы. Наконец, поля, предназначенные только для чтения, занимают предназначенные для них области памяти (что до определенной степени аналогично статическим объектам).

Проиллюстрируем основные свойства полей, предназначенных только для чтения, следующими примерами фрагментов программ на языке C#. Описание поля выглядит следующим образом:

```
readonly DateTime date;
```

Доступ к полю изнутри класса организуется по краткому имени объекта:

```
... value ... size ... date ...
```

Доступ к полю из других классов (на примере объекта `c` как конкретизации класса `C`) реализуется с указанием полного имени объекта:

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

Доступ к статическим полям и константам

Статические поля и константы принадлежат классу, а не объекту:

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

Доступ изнутри класса:

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

Доступ из других классов:

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


Комментарий к слайду

Рассмотрим подробнее особенности реализации механизмов доступа к статическим полям и константам в языке программирования C#.

Прежде всего, необходимо отметить следующее фундаментальное положение, характеризующее статические поля и константы. Поскольку изменения свойств объектов, динамических по своей природе и порожденных динамическими классами, не затрагивают статических полей и констант, последние принадлежат классам, а не объектам.

Проиллюстрируем это высказывание следующим примером фрагмента программы на языке C#:

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

Как видно из приведенного примера, при описании класса `Rectangle` со статическими полями `Color` и `scale` и динамическими полями `x`, `y`, `width` и `height`, статическими являются поля, инвариантные по отношению к классу, а динамическими – представляющие собой «неподвижную точку» относительно объекта.

Доступ к статическим полям изнутри класса по краткому имени и из других классов по полному имени соответственно реализуется по аналогии с полями только для чтения:

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

Доступ к методам в языке C# (1)

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


Комментарий к слайду

Продолжим иллюстрацию механизмов реализации концепции инкапсуляции в языке программирования C# следующим примером фрагмента программы:

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

Приведенный пример представляет собой описание класса `C`, содержащего целочисленные поля `sum` и `n`, а также методы `Add` и `Mean` для суммирования и вычисления среднего значения, реализованные в форме процедуры и функции соответственно.

Заметим, что методы `Add` и `Mean` класса `C` описаны как общедоступные и, следовательно, могут быть доступны из любого места программного проекта при указании полного имени, тогда как поля `sum` и `n`, в силу умолчаний, принятых в языке программирования C#, являются доступными локально (`private`).

Заметим попутно, что функция (в терминологии языка C#) отличается от процедуры тем, что обязательно возвращает значение. Для определения процедуры, не возвращающей значения, в C# используется зарезервированное слово `void`.

Доступ к методам в языке C# (2)

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

Проиллюстрируем сказанное об областях видимости объектов языка программирования C# примерами фрагментов программ.

Предположим, что в предыдущем примере, описывающем общедоступный класс `C`, содержащего целочисленные поля `sum` и `n`, а также методы `Add` и `Mean`, необходимо организовать доступ к элементам класса изнутри, а также из других классов.

Очевидно, что в силу общедоступности реализации класса `C` такое задание принципиально осуществимо.

При реализации доступа к элементам класса `C` изнутри данного класса достаточно указать краткие имена объектов:

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

В то же время, при реализации механизма доступа к элементам класса `C` из других классов (внешних по отношению к данному) необходимо указывать полные имена объектов:

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

Доступ к статическим методам 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

Манипулирование статическими полями и методами в языке программирования C# осуществляется вполне аналогично рассмотренным выше случаям статических полей и констант.

Например, в случае, если класс, содержащий статические элементы данных, определяется посредством следующего описания:

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

доступ к элементам данных изнутри класса осуществляется посредством указания краткого имени объекта класса, например:

```
        ResetColor();
```

а доступ к элементам данных из других классов – посредством указания полного имени объекта класса, например:

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

Преимущества использования инкапсуляции:

1. Унификация моделирования предметной области
2. Прямолинейный подход к моделированию предметной области
3. Гибкое управление уровнем абстракции данных (и метаданных)
4. Безопасность и независимость от пользователя результирующего кода

Комментарий к слайду

По результатам исследования концепции инкапсуляции и ее применения в языках объектно-ориентированного программирования можно сделать следующие выводы о принципиальных преимуществах рассмотренной концепции.

Прежде всего, реализация концепции инкапсуляции приводит к унификации представления, а, следовательно, и моделирования сколь угодно сложных предметных областей.

Кроме того, механизм инкапсуляции обеспечивает четкий, недвусмысленный, прямолинейный подход к моделированию предметной области за счет объединения объектов с данными.

Именно в силу последнего обстоятельства становится принципиально возможной реализация гибкого управления уровнем абстракции как для данных, так и для метаданных.

Наконец, концепция инкапсуляции гарантирует более высокую степень безопасности результирующего кода и его защищенности от несанкционированных или нецеленаправленных действий пользователя.

Библиография

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. Date C.J. Encapsulation is a Red Herring. DataBase Programming and Design On-LineFrei, Sept. 1998. (`www.dbpd.com/9809date.htm`)
4. Troelsen A. C# and the .NET platform (2nd ed.).- APress, 2003, 1200 p.p.
5. Liberty J. Programming C# (2nd ed.).- O'Reilly, 2002, 656 p.p.

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 инкапсуляция. 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.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.Date C.J. Encapsulation is a Red Herring. DataBase Programming and Design On-LineFrei, Sept. 1998. (`www.dbpd.com/9809date.htm`)

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

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

Кратко остановимся на источниках. Кратко остановимся на источниках. В работе [1] приведен наиболее полный анализ истории развития и особенностей языков программирования с классификацией по областям применения. В работах [2,4,5] рассмотрены теоретические проблемы и практические аспекты реализации конструкций в языках программирования, прежде всего, в языке C#.NET, изучение которого составляет основу курса. Работа [3] представляет собой исследование концепции инкапсуляции в современном программировании.