Основы программирования

Наследование

Поля и конструкторы

Особенности private полей

Дочерний класс использует структуру родительского класса, то есть наследует все его элементы, и открытые (`public`), и закрытые (`private`).

В теле дочернего класса нет непосредственного доступа к закрытым (`private`) полям, унаследованным от родителя (реализация инкапсуляции).

На этапе исполнения объект дочернего типа представляет собой единое целое с унаследованной родительской частью.

Базовый пример. Класс Person

```
class Person   {

private string name;

… // и еще конструктор

public string Name  {
      get { return name; }
      set { name = value; }   }

public void Display()   {
      Console.WriteLine(Name);  }

}
```

Базовый пример. Класс Employee

```
class Employee: Person   {
…
}
```

Наследование реализует отношение `is-a` (является).

То есть объект класса `Employee` также является объектом класса `Person`.

Базовый пример. Класс Program

```
class Program   {

static void Main(string[] args)  {
      Person p = new Person("Иван");
      p.Display();
      p = new Employee("Петр");

      //и так:
      //Person p1 = new Employee();

      p1.Display();
      Console.Read(); }
}
```

Наследование от класса Object

Все классы наследуются от базового класса `Object`.

Классы `Person` и `Employee` кроме своих собственных методов, также будут иметь и методы класса `Object`:

- `ToString`(),
- `Equals`(),
- `GetHashCode`(),
- `GetType`().

Ограничение на наследование 1

- Не поддерживается множественное наследование.
- Тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим.

  То есть, если базовый класс у нас имеет тип доступа `internal`, то производный класс может иметь тип доступа `internal` или `private`, но не `public`.

Ограничение на наследование 2

- Если базовый и производный классы находятся в разных сборках (проектах), то в этом случае производный класс может наследовать только от класса, который имеет модификатор `public`.
- Класс, объявленный с модификатором `sealed`, нельзя наследовать и создавать производные классы.

  Например: `sealed class Admin { }`
- Нельзя наследовать от статического класса.

Доступ с полям базового класса

Какой пример не будет работать и почему?

1

```
class Employee: Person
{
  public void DisplayPN()
  {
    Console.WriteLine(name);
  }
}
```

2

```
class Employee: Person
{
  public void DisplayEN()
  {
    Console.WriteLine(name);
  }
}
```

Модификаторы доступа бывают:

- **public**: доступен из любого места в коде, а также из других программ и сборок.
- **private**: доступен только из кода в том же классе или контексте.
- **protected**: доступен из любого места в текущем классе или в производных классах, в том числе и в других сборках.
- **internal**: доступны из любого места кода в той же сборке.
- **protected internal**: …
- **private protected**: …

Доступ к полям родительского класса

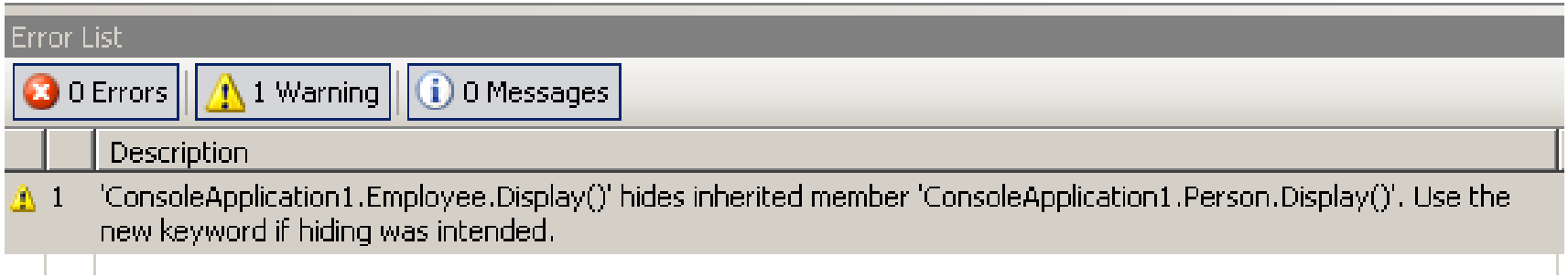
Производный класс может иметь доступ только к тем членам базового класса, которые определены с модификаторами

- **private protected** (если базовый и производный класс находятся в одной сборке),
- **public internal** (если базовый и производный класс находятся в одной сборке),
- **protected** и
- **protected internal**.

Базовый пример. Класс Employee

```
class Employee: Person   {
public void Display()      {
Console.WriteLine("Имя сотрудника" + Name); }
}
```

Предупреждение:



Фрагмент класса Program

```
Person p = new Person ("Иван");
p.Display();
p = new Employee ("Петр", "Яндекс");
p.Display();
```

```
Иван
Петр
```

```
Person p = new Person("Иван");
p.Display();
Employee emp = new Employee("Петр", "Яндекс");
emp.Display();
```

```
Иван
Имя сотрудника Петр
```

Класс Person. Конструктор

```
class Person   {

private string name;

public Person(string name)  {
    Name = name;      }

public string Name  {
      get { return name; }
      set { name = value; }    }
public void Display()   {
      Console.WriteLine(Name);  }

}
```

Класс Employee. Конструктор

```
class Employee: Person   {

private string company;

public string Company {
       get { return company; }
       set { company = value; }

public Employee(string name, string company): base(name)  {
        Company = company;  }

public void Display()     {
Console.WriteLine("Имя сотрудника:" + Name); }

}
```

Класс Program

```
class Program   {

static void Main(string[] args)  {

Person p = new Person("Иван");
p.Display();
Employee emp = new Employee("Петр","Яндекс");
emp.Display();
Console.ReadLine();
}

}
```

(НЕ !) наследование конструкторов

При наследовании конструкторы не наследуются, а вызываются.

Если в базовом классе не определен конструктор по умолчанию без параметров, а только конструкторы с параметрами (как в случае с базовым классом `Person`), то в производном классе мы обязательно должны вызвать один из этих конструкторов через ключевое слово `base`.

...

Если в классе `Employee` убрать определение конструктора, то будет ошибка, так как класс `Employee` не соответствует классу `Person`, а именно не вызывает конструктор базового класса.

Если в классе `Employee` добавить конструктор, который будет устанавливать значения тех же полей, например,

```
public Employee(string name, string company) {
    Name = name;
    Company = company; }
```

то ошибка все равно будет.

Решение 1

Явным образом вызвать конструктор класса `Person`:

```
   public Employee(string name, string
   company): base(name)
{
      Company = company;
}
```

Решение 2

Определить в базовом классе конструктор без параметров:

```
class Person            {
    …
    public Person()     {
      Name = "Tom";
      Console.WriteLine("Вызов конструктора без
параметров");
    }
}
```

Решение 2

Тогда конструктор все равно вызовет конструктор без параметров базового класса:

```
public Employee(string company){
    Company = company;
}
```

То есть эквивалентен:

```
public Employee(string company): base()     {
    Company = company;
}
```

Порядок вызова конструкторов

При вызове конструктора класса сначала отрабатывают конструкторы базовых классов и только затем конструкторы производных.

Порядок вызова конструкторов

```
class Person     {

private string name; int age;

public Person(string name)     {
this.name = name;
Console.WriteLine("Person(string name)"); }

public Person(string name, int age): this(name)          {
this.age = age;
Console.WriteLine("Person(string name, int age)");     }
}
```

Порядок вызова конструкторов

```
class Employee : Person   {

private string company;

public Employee(string name, int age, string company) :
base(name, age)  {
this.company = company;
Console.WriteLine("Employee(string name, int age,
string company)");     }
}
```

Порядок вызова конструкторов

При создании объекта Employee:

```
Employee emp = new Employee("Петр", 22, "Яндекс");
```

Получим:

```
Person(string name)
Person(string name, int age)
Employee(string name, int age, string company)
```

Порядок вызова конструкторов

1. `Employee(string name, int age, string company)` делегирует выполнение конструктору `Person(string name, int age)`.
2. `Person(string name, int age)` (пока не выполняется) передает выполнение конструктору `Person(string name)`.
3. `Person(string name)` передает выполнение конструктору класса `System.Object`.
4. Выполняется `System.Object.Object()`, выполнение возвращается конструктору `Person(string name)`.

Порядок вызова конструкторов

4. Выполняется конструктор `Person(string name)`, выполнение возвращается конструктору `Person(string name, int age)`.
6. Выполняется конструктор `Person(string name, int age)`, выполнение возвращается конструктору `Employee(string name, int age, string company)`.
7. Выполняется конструктор `Employee(string name, int age, string company)`
   В итоге создается объект `Employee`.