Интерфейсы

Jelena Švartsman

- Понятие интерфейса чем-то похоже на абстрактный класс.
- Интерфейс — это полностью абстрактный класс, не содержащий никаких полей, кроме констант.

Терминология.

Класс *наследует* другой класс, **но,**

класс *удовлетворяет* интерфейсу,

класс *реализует, выполняет* интерфейс.

Существует, однако, серьезное отличие интерфейсов от классов вообще и от абстрактных классов, в частности. Интерфейсы допускают **множественное наследование**. Т.е. один класс может удовлетворять нескольким интерфейсам сразу. Это связано с тем, что интерфейсы не порождают проблем с множественным наследованием, так как они не содержат полей.

Синтаксис:

```
interface IXXX
{  . . .
    int f(String s);
}
```

Это описание интерфейса IXXX.
Внутри скобок могут находиться только описания методов (без реализации) и описания свойств.
В данном случае интерфейс IXXX содержит, в частности метод f(...).

```
public class R : IXXX
{    . . .
}
```

Класс R реализует интерфейсы IXXX. Внутри класса, реализующего интерфейс, должны быть реализованы все методы, описанные в этом интерфейсе.

Поскольку IXXX имеет метод f(...), то в классе R он должен быть реализован:

```
public class R : IXXX
{       . . .
        public int f(String s)
        {
                        ...
        }
        ...
}
```

Обратите внимание, что в интерфейсе f(...) описан без описателя public,
а в классе R с описателем **public**.

Все методы интерфейса по умолчанию считаются public.
В классе R мы обязаны модификатор доступа public использовать явно.

Еще одним общим моментом интерфейсов и абстрактных классов является то, что хотя и нельзя создавать объекты интерфейсов, но можно описывать переменные типа интерфейсов.

Интерфейсы

```
interface IExample{

    bool Done();
}
class MyClass: IExample{
    public bool Done(){
    ...
    }
}
```

- При объявлении интерфейса нельзя указывать модификаторы доступа!

Интерфейсы - использование

```
interface I1{
    void Foo();
}
class MyClass: I1{
    public void Foo(){...};
}
...
public static void Main(){
    MyClass cl1 = new MyClass();
    ((I1)cl1).Foo(); //1-ый способ
    I1 icl1 = (I1)cl1;
    icl1.Foo(); //2-ой способ
}
```

«Множественное наследование»

```
interface I1{
   void Foo();
}
interface I2{
   bool Bar(int var1);
}
class MyClass: I1, I2{
   public void Foo() {…};
   public bool Bar() {…};
}
```

«Множественное наследование» - потенциальные проблемы

- Проблема:

```
interface I1{
  void Foo();
}
interface I2{
  void Foo();
}
class MyClass: I1, I2{
  public void Foo() {…};
}
```

- Решение:

```
interface I1{
  void Foo();
}
interface I2{
  void Foo();
}
class MyClass: I1, I2{
  public void Foo() {…};
  public void I1.Foo() {…};
  public void I2.Foo() {…};
}
```

Интерфейсы - as

```
interface I1{
    void Foo();
}
class MyClass: I1{
    public void Foo() {…};
}
…
public static void Main(){
    MyClass cl1 = new MyClass();
    I1 icl1 = cl1 as I1;
    if (icl1 != null) { icl1.Foo(); }
}
```

Комбинирование интерфейсов

```
public interface IDrag{
    bool Drag();
}
public interface IDrop{
    bool Drop();
}
public interface IDragNDrop: IDrag, IDrop{
}
```

Интерфейсы широко используются при написании различных стандартов.

Стандарт определяет, в частности, набор каких-то интерфейсов. И, кроме того, он содержит вербальное описание семантики этих интерфейсов. Прикладные разработчики могут писать программы, используя интерфейсы стандарта.

А фирмы-разработчики могут разрабатывать конкретные реализации этих стандартов.

При внедрении (deployment) прикладного программного обеспечения можно взять продукт любой фирмы-разработчика, реализующий данный стандарт (на практике, конечно, все несколько сложнее).