Узагальнені типи делегатів

Тип делегата може містити параметри узагальненого типу.

Наприклад:

```
public delegate T Transformer<T> (T arg);
```

Так можна описати узагальнений службовий метод, який буде працювати з будь-яким типом.

```
public class Util{
public static void Transform<T> (T[] values, Transformer<T>
t){
for (int i = 0; i < values.Length; i++)
values[i] = t (values[i]);}}
class Test{
static void Main(){
int[] values = { 1, 2, 3 };
Util.Transform (values, Square);// Прив'язатися до  Square
foreach (int i in values)
Console.Write (i + " ");//1 4 9
}
static int Square (int x) { return x * x; }
}
```

Делегати Func і Action

Дозволяють працювати з методами, що повертають будь-який тип і мають будь-яку (розумну) кількість аргументів.

Func і Action визначені в просторі імен System.

```
delegate TResult Func <out TResult> ();
delegate TResult Func <in Tf out TResult> (T arg) ;
delegate TResult Func <in Tlf in T2, out TResult> (Tl
argl, T2 arg2);
. . . і т.д. до T16
delegate void Action () ;
delegate void Action <in T> (T arg) ;
delegate void Action <in Tlf in T2> (Tl argl, T2 arg2)
;
. . . і т.д. до T16
```

Заміна Transformer на Func у попередньому прикладі, який приймає 1 аргумент типу T і повертає значення цього ж типу.

```
public static void Transform<T> (T[] values,
Func<T,T> transformer)
{
for (int i = 0; i < values.Length; i++)
values[i] = transformer (values[i]);
}
```

Делегати Func і Action не покривають лише практичні сценарії, пов'язані з параметрами ref/out і параметрами покажчиків.

Порівняння делегатів та інтерфейсів

```
public interface ITransformer{
int Transform (int x) ;}
public class Util{
public static void TransformAll (int[] values,
ITransformer t){
for (int i = 0; i < values.Length; i++)
values[i] =t.Transform (values[i]);}}
class Squarer : ITransformer{
public int Transform (int x) { return x * x; }}
static void Main(){
int[] values = { 1, 2, 3 };
Util.TransformAll (values, new Squarer());
foreach (int i in values)
Console.WriteLine (i);}
```

Розв'язок на основі делегатів може виявитися вдалішим, ніж розв'язок на основі інтерфейсів, якщо виконується одна або більше з наступних умов:

- інтерфейс визначає лише один метод;
- потрібна можливість групового виклику;
- той, хто підписується, повинен реалізувати інтерфейс кілька разів.

В прикладі ITransformer груповий виклик не потрібний. Проте, інтерфейс визначає лише один метод. Більш того, підписчикові може потрібно реалізувати ITransformer кілька разів, щоб підтримувати різні трансформації, такі як піднесення до квадрату або до кубу. За допомогою інтерфейсів нам доведеться писати окремий тип для кожної трансформації, оскільки Test може реалізувати ITransformer лише один раз. В результаті виходить досить громіздкий код:

```
class Squarer : ITransformer
{
public int Transform (int x) { return x * x; }
}
class Cuber : ITransformer
{
public int Transform (int x) {return x * x * x; }
static void Main()
{
int[] values = { 1, 2, 3 };
Util.TransformAll (values, new Cuber());
foreach (int i in values)
Console.WriteLine (i);
}
```

Сумісність делегатів

Усі типи делегатів несумісні один з одним, навіть якщо їх сигнатури виглядають однаково:

```
delegate void D1();
delegate void D2();
D1 d1 = Method1;
D2 d2 = d1; // Помилка на етапі компіляції
```

Але наступне дозволено:

```
D2 d2 = new D2 (d1) ;
```

Екземпляри делегатів вважаються рівними, якщо вони мають один і той же цільовий метод:

```
delegate void D();
D d1 = Method1;
D d2 = Method1;
Console.WriteLine (d1 == d2);// true
```

Групові делегати вважаються рівними, якщо вони посилаються на одні і ті ж методи в ***однаковому порядку***.

Коваріантність та контрваріантність

class Person{…}

class Student : Person {…}

Коваріантність

```
class People
{
        Person GetPersons();
}
class CoursePeople : People
{
        Student GetPersons();
}
```

Дотримання контракта базового класу

```
class People
{
        void SetPersons(Person p);
}
class CoursePeople : People
{
        void SetPersons(Student p);
}
```

Порушення контракту базового класу
Порушення принципа ООП

Можлива тільки у вихідних параметрах

Контрваріантність

```
class ClassPeople
{
        Student GetStudents();
}
class TodayPeople : ClassPeople
{
        Person GetStudents();
}
```

Порушення контракта базового класу
Порушення принципа ООП

```
class ClassPeople
{
        void SetStudents(Student p);
}
class TodayPeople : ClassPeople
{
        void SetStudents (Person p);
}
```

Дотримання контракта базового класу

Можлива тільки у вхідних параметрах

Коваріантність

Коваріантність – приведення часткового (окремого) до загального.

В термінах ООП:

Там де потрібний базовий тип можна присвоїти екземпляр типу нащадка

Делегати коваріантні за типом, що повертається.

Під час виклику методу можна передавати аргументи, що відносяться до специфічніших типів, ніж визначено для параметрів цього методу. Це звичайна поліморфна поведінка. З тієї ж самої причини делегат може мати специфічніші типи параметрів, ніж його цільовий метод. Це називається **контрваріантністю**.

Наприклад:

```
delegate void StringAction (string s);
class Test{
static void Main(){
StringAction sa = new StringAction
(ActOnObject);
sa("hello");
}
static void ActOnObject (object o){
Console.WriteLine (o);//hello
}
}
```

Як і з варіантністю параметрів типу, делегати підтримують варіантність лише для *вказівникових перетворень*. Делегат просто викликає метод від імені когось іншого. У цьому випадку Stringaction викликається з аргументом типу string. Коли аргумент потім передається цільовому методу, він неявно приводиться вгору до object.

Стандартний шаблон подій спроектований для допомоги в передачі контрваріантності через використання загального базового класу Eventargs. Наприклад, можна мати єдиний метод, що викликається двома різними делегатами, причому один з них передає MouseEventArgs, а інший — KeyEventArgs.

Сумісність типів, що повертаються

В результаті виклику методу можна отримати назад тип, який є специфічніший, ніж запитаний. Це звичайна поліморфна поведінка. З тієї ж самої причини цільовий метод делегата може повертати специфічніший тип, ніж описаний делегатом. Це називається **коваріантністю**.

Наприклад:

```
delegate object ObjectRetriever();
class Test{
static void Main(){
ObjectRetriever o = new ObjectRetriever (RetriveString)
;
object result = o();
Console.WriteLine (result); // hello
}
static string RetriveString() { return "hello"; }}
```

Делегат ObjectRetriever очікує отримати назад object, але може бути отриманий також і підклас object; типи делегатів, що повертаються, є коваріантними.

Варіантність параметрів типу узагальненого делегата

При визначенні узагальненого типу делегата рекомендується робити таким чином:

- позначати параметр типу, що використовується лише для значення, що повертається, як коваріантний (out);
- позначати будь-який параметр типу, що використовується лише для аргументу, як контрваріантний (in).

Це дає можливість перетворенням працювати природним чином, дотримуючи спадкування між типами.

У прикладі делегат (визначений в просторі імен System) підтримує коваріантність:

```
delegate TResult Func<out TResult>();
```

Це дозволяє наступне:

```
Func<string> x = . . . ;
Func<object> y = x;
```

Показаний у прикладі нижче делегат (визначений в просторі імен System) підтримує контрваріантність:

```
delegate void Action<in T> (T arg) ;
```

Це дозволяє наступне:

```
Action<object> x = . . . ;
Action<string> y = x;
```