Шаблонни (типизирани) класове, интерфейси и методи (templates, generics)

Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

Съдържание

1. Защо въобще са ни нужни?
2. Синтаксис на шаблонните класове
3. Предимства от използването им
4. Видимост на параметъра за типа
5. Разгъване на шаблонен клас
6. Шаблонни интерфейси
7. Шаблонни методи
8. Използване на default
9. Ограничители на шаблонни класове

Задача: кутия за всичко

- Създайте клас **Box**, който да може да съхранява **всякакви неща** (но всички да са от един и същи тип)
  - Добавянето трябва да поставя новото най-отгоре
  - Премахването да взима най-горния елемент
- Трябва да има два публични метода:
  - `public void Add(element item);`
  - `public element Remove();`
- Добре е да има и проверка дали *item* е от правилния тип

Реализациите за различни типове данни ще са подобни

Какъв тип да поставим тук?

Шаблонни класове – предимства

- Позволява параметризиране на използваните типове данни:

```
Box<string> box =
    new Box<string>();
box.Add("one");
box.Add(2); // грешка
```

```
Box<int> box =
    new Box<int>();
box.Add(1);
box.Add("2"); // грешка
```

- Добавя проверка на типа (Type Safety) на клиента
- Осигурява мощен начин за повторно използване на кода

```
List<int> numbers = new List<int>();
List<Person> people = new List<Person>();
```

Шаблонни класове - синтаксис

- Дефинира се с <Тип 1, Тип 2 … и т.н.>

```
class List<T> {
    …
}
```

- Може да има множество параметри за типове

```
class Dictionary<TKey, TValue> {
    …
}
```

Видимост на параметъра за типа

- Може да бъде използван навсякъде в класа

```
class Box<T> {
    private T[] data;
    public T Top { get {…} }
    public void Add (T item) {…}
    public T Remove () {…}
}
```

Решение: кутия с T

- В случая T е като параметър, определящ типа на данните

```
public class Box<T> {
  public T[] data;
  public int Count { get;
                     private set; }

  public Box(int size) {
    this.data = new T[size];
    this.Count = 0;
  }
```

```
  public void Add(T item) {
    data[Count] = item;
    Count++;
  }

  public T Remove() {
    Count--;
    return data[Count];
  }
}
```

Разгъване на шаблонен клас

- Можете да го разширите с конкретен клас

```
class BoxOfPickles : Box<Pickle> {
    …
}
```

```
BoxOfPickles jar = new BoxOfPickles();
jar.Add(new Pickle());
jar.Add(new Vegetable()); // Error
```

Шаблонни интерфейси

- Подобни са на шаблонните класове

```
interface IBox<T> {
    void Add (T element);
    …
}
```

```
class MyList : IBox<MyClass> {…}
```

```
class MyList<T> : IBox<T> {…}
```

Шаблонни методи

- Може да имат какъв да е вход и връщан резултат

```
public List<T> createList(T item, int count) {
  List<T> list = new List<T>();
  for (int i = 0; i < count; i++) {
    list.Add(item);
  }
  return list;
}
```

- Статичните методи също могат да бъдат шаблонни
- Конструкторите и свойствата – не

Задача: Създател на шаблонен масив

- Създайте клас **`ArrayCreator`** с един-единствен метод:
  - **`static T[] Create(int length, T item)`**
- Той трябва да връща масив
  - С указаната дължина
  - Всички елементи трябва да бъдат от типа, подаден като параметър

Решение: Създател на шаблонен масив

```
public static class ArrayCreator
{
    public static T[] Create<T>(int lenght, T item)
    {
        T[] array = new T[length];
        return array;
    }
}
```

Задача: Универсална везна

- Създайте клас **`Scale<T>`** който:
  - Съдържа два елемента: left и right
  - Получава елементите чрез своя единствен конструктор:
    - Scale(T left, T right)
  - Има метод: **`T GetHeavier()`**
- По-големият от двата елемента е по-тежък
- **Проблем:** Ако елементите са равни, какво да върне?
  - Ако T е референтен тип, трябва да е null, а ако е числов - 0

Оператор default(T)

- връща подразбиращата се стойност за конкретния тип:
  - за референтни типове: null
  - за числови типове: 0
  - за булев тип: false, за символен: '\0' и т.н.

- Т.е. нашата везна трябва да връща **`default`**, ако елементите са равни

Решение: Универсална везна

```
public class Scale<T> where T : IComparable<T>
{
  private T left;
  private T right;

  public Scale(T left, T right)
  {
    this.left = left;
    this.right = right;
  }
  //TODO: продължава на следващия слайд
}
```

Ограничител на типа

Решение: Универсална везна (2)

```
public T GetHavier()
{
  if (left.CompareTo(right) > 0)
  { return left; }
  else if (left.CompareTo(right) < 0)
  { return right; }
  return default(T);
}
```

Ограничаване до референтен тип

- Ограничителите се представят в C# с ключовата дума **where**
- Указване, че T трябва да е референтен тип

```
public void MyMethod< T >()
       where T : class
{
   ...
}
```

- **class** тук е ключова дума и трябва да е с малки букви

Ограничаване до примитивен тип

- Указване, че T трябва да е примитивен тип

```
public void MyMethod< T >()
        where T : struct
{
   ...
}
```

- `struct` тук е ключова дума и трябва да е с малки букви

Ограничаване до конструктор

- Указване, че T трябва да е конструктор

```
public void MyMethod< T >()
       where T : new ()
{
   ...
}
```

- Само default конструктор може да бъде използван
- Параметризиран конструктор ще доведе до грешка при компилиране

Ограничаване до даден базов клас

- Указване на даден базов клас като ограничение

```
public void MyMethod< T >()
       where T : BaseClass
{
   ...
}
```

- Типът параметър трябва да е от указания базов клас или да е негов наследник

Ограничаване до шаблонен базов клас

- Указване на шаблонен базов клас като ограничение

```
public void MyMethod< T, U >()
       where T : U
{
   ...
}
```

- Типът параметър за T трябва да е от класа-параметър U или да е негов наследник

Комбиниране на ограничителите

- Указване на няколко базови класа и конструктор като ограничение

```
public void MyMethod< T >()
  where T : IComparable, MyBaseClass, new ()
{
  ...
}
```

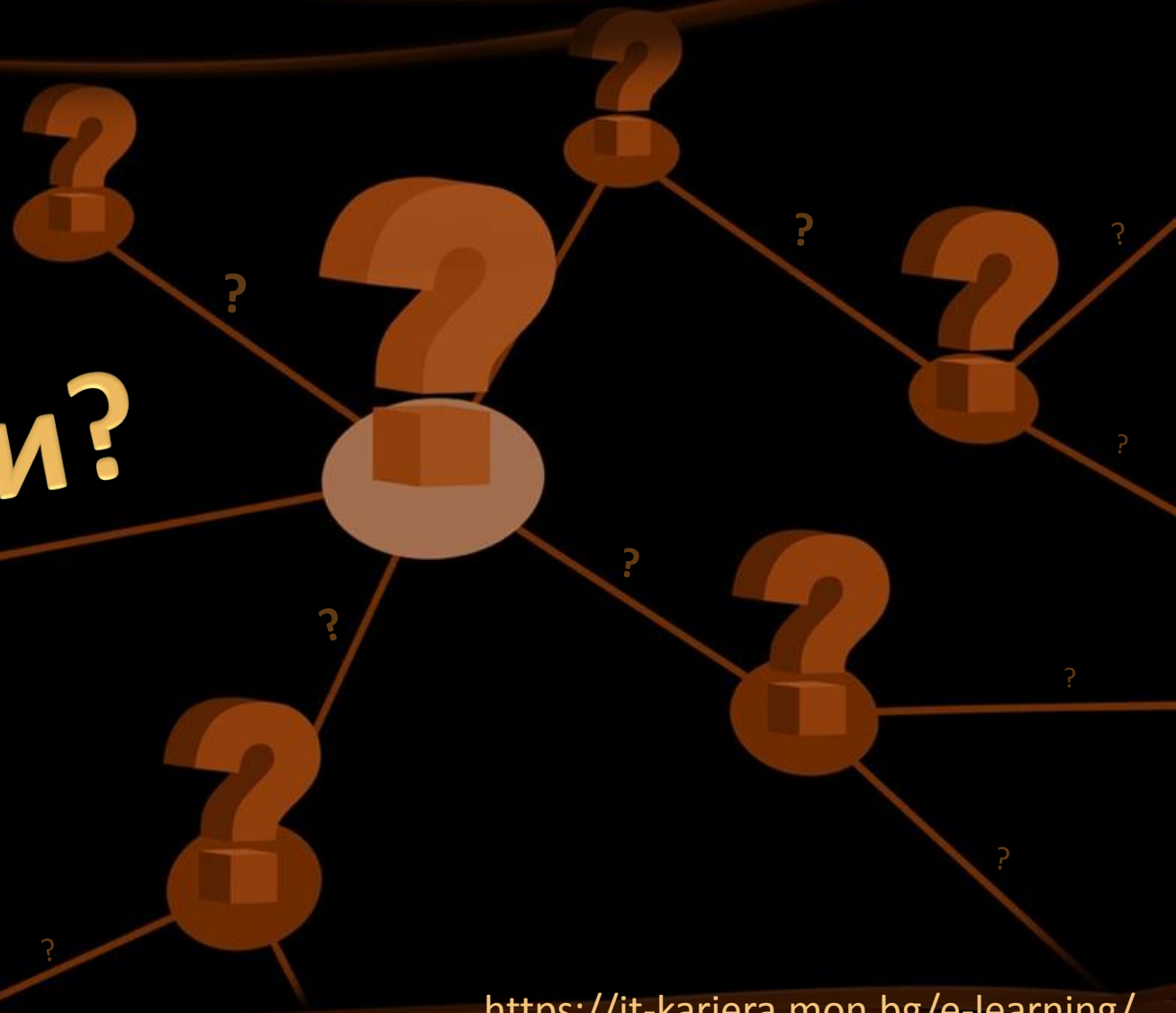
- Невалидни комбинации от ограничители: `class` и `struct`

Обобщение

- Код със шаблонни класове и интерфейси позволява повторна употреба и проверка на типа
  - Параметърът за типа може да се използва навсякъде в описанието на класа
- Шаблонните методи са по-универсални
- Ограничителите помагат да се ограничи типовия параметър
- Default връща подразбиращата се стойност на параметъра за типа

Шаблонни класове, интерфейси и методи

Въпроси?



https://it-kariera.mon.bg/e-learning/

Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"

- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Основи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA