ОБОБЩЕНИЯ

Boxing & unboxing

Упаковка предполагает преобразование объекта значимого типа к типу object. При упаковке общеязыковая среда CLR обертывает значение в объект типа System.Object и сохраняет его в куче.

Распаковка предполагает преобразование объекта типа object к значимому типу.

Упаковка и распаковка ведут к снижению производительности, так как системе надо осуществить необходимые преобразования.

Type safety

Типобезопасность языка программирования означает безопасность/надёжность его системы типов.

Система типов называется безопасной (safe) или надежной (sound), если в программах, прошедших проверку согласования типов, (well-typed programs или well-formed programs) исключена возможность возникновения ошибок согласования типов во время выполнения.

Ошибка согласования типов или ошибка типизации (type error) представляет собой несогласованность типов компонентов выражений в программе, например попытку использовать целое число в роли функции.

Generics

Обобщения (универсальные типы и методы) позволяют разрабатывать классы и методы, которые откладывают спецификацию одного или нескольких типов до тех пор, пока класс или метод не будут объявлены и экземпляры не будут создаваться клиентским кодом.

Пример

```
public class Item<T>
{
    0 references
    public string Name { get; set; }
    0 references
    public T Parameters { get; set; }
}
```

```
var table = new Item<int[]>()
{
    Name = "Table",
    Parameters = new int[] { 30, 65, 70 }
};

var chair = new Item<string>()
{
    Name = "Chair",
    Parameters = "Brown"
};
```

Generics

Достоинства дженериков:

- возможность многократного использования
- типобезопасность
- эффективность

Универсальные типы наиболее часто используются с коллекциями и методами, которые выполняют с ними операции.

Generics

- Использование дженериков позволяет получить максимально широкие возможности многократного использования кода, обеспечения безопасности типов и повышения производительности.
- Чаще всего универсальные шаблоны используются для создания классов коллекций.
- Библиотека классов .NET содержит несколько универсальных классов коллекций в пространстве имен System.Collections.Generic.
- .NETпозволяет создавать универсальные интерфейсы, классы, методы, события и делегаты.
- Универсальные классы можно ограничить, чтобы они разрешали доступ к методам только для определенных типов данных.
- Сведения о типах, используемых в универсальном типе данных, можно получить во время выполнения с помощью рефлексии.

Использование нескольких дженериков

```
public class User<T, K>
{
    1 reference
    public T Id { get; private set; }
    1 reference
    public string Login { get; private set; }
    1 reference
    public K Password { get; set; }

    1 reference
    public User() { }

    1 reference
    public User(T id, string login, K password)...
}
```

```
var user1 = new User<int, string>(1930, "superproger", "qwerty");
var user2 = new User<int[], char>();
```

Ограничения дженериков

В качестве ограничений можно использовать следующие типы:

- классы
- интерфейсы
- class - универсальный параметр должен представлять класс
- struct - универсальный параметр должен представлять структуру
- new() - универсальный параметр должен представлять тип, который имеет общедоступный конструктор без параметров

Пример

```
public class Item
{
    0 references
    public string Name { get; set; }
    0 references
    public int Price { get; set; }
}

0 references
public class Food : Item
{
    0 references
    public int Weight { get; set; }
}
```

```
public class Shop<TItem> where TItem : Item { }

0 references
public class Busket<T> where T : class, new() { }
```

Пример

```
public class Node<T> where T : IComparable<T>
{
    0 references
    public Node<T> Left { get; set; }

    0 references
    public Node<T> Right { get; set; }

    0 references
    public T Data { get; set; }

    0 references
    public int CompareTo(Node<T> other)...
}
```

```
public class Tree<T> where T : IComparable<T>
{
    0 references
    public Node<T> Root { get; set; }

    4 references
    public void Add(T element) { }

    0 references
    public void Remove(T element) { }
}
```

Пример

```
var tree = new Tree<int>(); //Int32
tree.Add(25); //Int32
tree.Add("123"); //string
tree.Add(Convert.ToInt32(new object())); //object
tree.Add((Int16)17); //Int16
```

Наследование дженериков

Существуют следующие варианты наследования:

- создание класса-наследника, который типизирован тем же типом, что и базовый
- создание обычного необобщенного класса-наследника. В этом случае при наследовании у базового класса надо явным образом определить используемый тип
- типизацию производного класса параметром совсем другого типа, отличного от универсального параметра в базовом классе. В этом случае для базового класса также надо указать используемый тип
- использование универсального параметра из базового класса с применением своих параметров

Пример

```
//1st case
0 references
class Furniture<T> : Item<T> { }

//2nd case
0 references
class Furniture : Item<int[]> { }

//3rd case
0 references
class Food<T> : Item<int[]> { }

//4th case
0 references
class Food<T, K> : Item<T> { }
```