

Ambientes Virtuais de Execução

Genericos

Genéricos

► Objectivo

- Independência de tipos na definição de:
 - Tipos de dados (classes, estruturas, interfaces, *delegates*)
 - Algoritmos (métodos)
- Sem perder:
 - Robustez
 - Desempenho
 - Expressividade

► Exemplo

- Realizar uma fila homogénea de objectos, independente do tipo de objecto
- Definição: **class Queue<T>{...}**
- Utilização: **Queue<int> qi; Queue<string> qs;**



Genéricos

- ▶ Exemplo: considere uma classe Box que armazena um objecto qualquer.

```
public class MyBox{  
    private Object element;  
    public void add(Object element){  
        this.element = element; }  
    public Object get(){  
        return element; }  
}
```

- ▶ Como os métodos desta classe recebem ou retornam **Object**, pode ser passado um parâmetro de qualquer tipo.
- ▶ Como saber à priori qual é o tipo concreto de *element* que está agregado a um objecto do tipo MyBox?
 - ▶ Especificar num comentário não indica ao compilador o tipo de objecto.

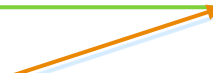
Genéricos

```
public class Teste1{  
    public static void Main(String[] args){  
        //Apenas colocar inteiros na Box  
        MyBox intBox = new MyBox();  
        intBox.add(10);  
        int someInt = (int) intBox.get();  
        Console.WriteLine(someInt);  
    }  
}
```



Genéricos

```
public class Teste2{  
    public static void Main(String[] args){  
        MyBox intBox = new MyBox();  
        //...  
        intBox.add("10");  
        // note que agora o tipo é String  
        // ...  
        int someInt = (int) intBox.get();  
        Console.WriteLine(someInt);  
    }  
}
```



Unhandled Exception: System.InvalidCastException:
Specified cast is not valid.
At Teste2.Main(String[] args)

**Exceção em
Runtime**

Genéricos

- ▶ Se a classe **MyBox** tivesse sido desenhada tendo em conta os genéricos, isto é, uma **classe genérica**, este engano teria sido apanhado em **compile time** em vez de a aplicação terminar anormalmente (em *run time*).

```
/*Versão genérica da classe Box.*/  
public class MyBox<T>{  
    private T t;  
    public void add(T t){  
        this.t = t; }  
    public T get(){  
        return t; }  
}
```

- ▶ Para referenciar esta classe genérica no nosso código, tem de ser realizada uma invocação de tipo genérica, que substitui T por um tipo concreto, tal como:
 - ▶ **MyBox<String> integerBox;**
 - ▶ **MyBox<int> intBox;**

Genéricos e Value Types

```
public class Teste1{
public static void Main(String[] args){
    MyBox intBox = new MyBox();
    intBox.add(10);    //box
    int someInt = (int) intBox.get(); //unbox
    Console.WriteLine(someInt);
}
}
```

```
public class Teste1{
public static void Main(String[] args){
    MyBox<int> intBox = new MyBox<int>();
    intBox.add(10);    //sem box
    int someInt = intBox.get(); //sem unbox
    Console.WriteLine(someInt); }
}
}
```

Genéricos

- ▶ **Type Safety**

- ▶ Permite criar colecções homogéneas, validadas em tempo de compilação

- ▶ **Aumento da legibilidade**

- ▶ O código não necessita de *cast*'s explícitos

- ▶ **Aumento de performance**

- ▶ Instâncias de tipo valor não precisam de ser boxed para serem guardadas em colecções genéricas.



Exemplo: *Lista* genérica

```
public class GenericList<T> {  
    // A classe Node é também genérica em T  
    private class Node {  
        private Node next;  
        private T data;  
        Node(T t) { next = null; data = t; }  
        public Node Next {  
            get { return next; } set { next = value; } }  
        public T Data {  
            get { return data; } set { data = value; } }  
    }  
  
    private Node head;  
    public GenericList() { head = null; }  
    public void AddHead(T t) {  
        Node n = new Node(t); n.Next = head; head = n;  
    }  
    // . . .  
}
```

Exemplo: *Stack* genérico

```
public class Stack<T> {  
    int sp = 0;  
    T[] items = new T[100];  
    public void Push(T item) { items[sp++] = item; }  
    public T Pop() { return items[--sp]; }  
}  
public class Example {  
    public static void Main() {  
  
        Stack<string> strStack = new Stack<string>(); // Stacks de tipos  
        Stack<int> intStack = new Stack<int>();       // distintos  
        string s;  
        int i;  
  
        strStack.Push("X");  
        intStack.Push(8);                               // sem box  
  
        s = strStack.Pop();                             // sem cast  
        i = intStack.Pop();                             // sem unbox  
  
        //intStack.Push("8");                          // não compila!  
    }  
}
```

+ Expressividade

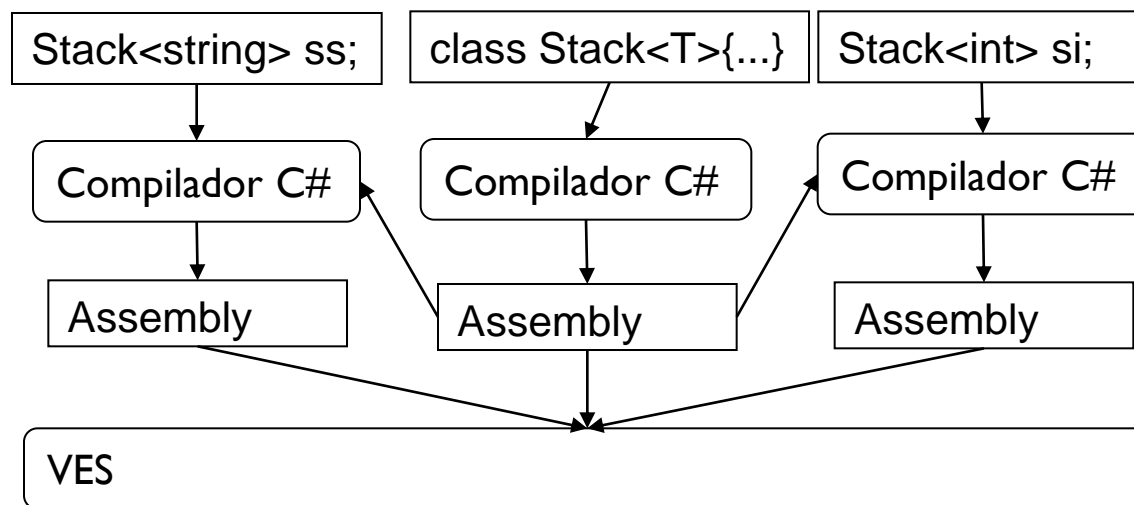
+ Desempenho

+ Robustez

Compilação de Genéricos

■ Genéricos

- ▶ O código genérico é compilado para IL, que fica com informação genérica de tipos
 - ▶ representação intermédia ainda é genérica
 - ▶ genérico é usável na forma compilada CIL
- ▶ O compilador não conhece a interface dos tipos que vão ser usados na instanciação do genérico
 - ▶ limita as acções realizáveis sobre objectos dos tipos-parâmetro



Genéricos em Java

- ▶ A compilação da classe genérica `Generic` é equivalente ao código da direita (um processo que se denomina de erasure, mais informação de metadata que indica que a classe originalmente era genérica e os correspondentes parâmetros)

```
class Generic<T> {  
    T value;  
    Generic(T value) {  
        this.value=value;  
    }  
  
    T getValue() {  
        return value;  
    }  
  
    void setValue(T t) {  
        value=t;  
    }  
}
```

Erasure

```
class GenericObj {  
    Object value;  
    public GenericObj(Object value) {  
        this.value=value;  
    }  
  
    Object getValue() {  
        return value;  
    }  
  
    void setValue(Object t) {  
        value=t;  
    }  
}
```

- ▶ Problemas:
 - ▶ Diferentes especializações partilham campos estáticos
 - ▶ Construções de objectos do tipo genérico
 - ▶ `extend` e `implements` de especializações do mesmo tipo ou interface genéricos



Genéricos – Code Explosion

- ▶ Quando um método que usa parâmetros do tipo genérico é compilado pelo JIT, o CLR
 - ▶ substitui o tipo genérico dos argumentos de cada método por cada tipo especificado, criando código nativo específico para operar para cada tipo de dados especificado
 - ▶ CLR fica com código nativo gerado para cada combinação método/tipo
 - ▶ Isto designa-se por **Code Explosion**.

Collecções genéricas e não genéricas na FCL

O namespace `System.Collections.Generic` contém classes e interfaces que definem colecções genéricas.

Colecção genérica	Colecção não genérica
<code>List<T></code>	<code>ArrayList</code>
<code>Dictionary<TKey,TValue></code>	<code>Hashtable</code>
<code>SortedDictionary<TKey,TValue></code>	<code>SortedList</code>
<code>Stack<T></code>	<code>Stack</code>
<code>Queue<T></code>	<code>Queue</code>
<code>LinkedList<T></code>	<code>(none)</code>
<code>SortedList<K,V></code>	<code>SortedList</code>

Interfaces genéricas e não genéricas na FCL

Interfaces Genéricas	Interfaces não genéricas
<code>IList<T></code>	<code>IList</code>
<code>IDictionary<Tkey,Tvalue></code>	<code>IDictionary</code>
<code>ICollection<T></code>	<code>ICollection</code>

Interface Genéricas	Interfaces não genéricas
<code>IEnumerator<T></code>	<code>IEnumerator</code>
<code>IEnumerable<T></code>	<code>IEnumerable</code>
<code>IComparer<T></code>	<code>IComparer</code>
<code>IComparable<T></code>	<code>IComparable</code>

System.Collections.Generic (interfaces e implementações)

