# Genéricos

1.

## Open e closed types

- List<T> → open type (contém parâmetros de tipo genérico)
  - ↳ não pode ser instanciado caso não se defina o T: é como se tratasse duma classe asstracta.
  - ↳ contudo, pode ser instanciado dentro doutra classe genérica no parâmetro T.

- List<int>, List<string> → closed types ou constructed types
  - ↳ argumento de tipo
  - ↳ já podem ser instanciados

- O AVE mantém estrutura RTTI com informações de cada open e closed type usados.

# Genéricos ≠ Templates



## Templates C++:

```
template < typename T >
class Stack {
    T[] arr; ...
    T pop() {
        T obj;
        ... obj.indexOf(1);
        --
    }
}
```

Posso invocar qualquer método sobre um objeto de tipo genérico, mesmo que não funcione (não aconselhável)

verificado pelo compilador em tempo de compilação

↓

Código da classe template colocado num ficheiro header (.h) para que o compilador conheça a interface dos parâmetros de tipo e assim gerar código em tempo de compilação.

```
#include <stack>

main:
    Stack<int> s1;
    Stack<string> s2;
```

→ compilador gera 2 tipos distintos e verifica correção do código.

Vantagens:

- Template meta-programming
  - pode-se realizar cálculos com os Templates sendo tudo verificado e expandido em tempo de compilação.

Problemas:

- "Code bloat" → ficheiro executável cresce à medida do nº de Templates instanciados.
- Templates distribuídos em source code e não já compilados.

# Genéricos Java



- "Type Erasure" de tipos genéricos

```
class Generic<T> {
    T value;
    Generic(T value){
      this.value = value;
    }
}
```

Erasure ⟹

```
class GenericObj {
    Object value;
    GenericObj(
      Object v) {
      this.value = v;
    }
}
```

- É feita verificação em tempo de compilação na declaração da classe (independentemente da utilização feita no Main)
- Não são suportados genéricos instanciados com tipos primitivos.

# Genéricos .Net



- Verificação em tempo de compilação na declaração
- Tipo genérico é compilado para IL sem "type erasure"
  - ↳ representação intermédia ainda é genérica

  vantagem: genéricos podem ser distribuídos já compilados
- Tal como no Java, o compilador não conhece a interface dos tipos que vão ser usados na instância do genérico

  Ex: `class B<T1, T2> { ... m(T1 a) { a.` ↳ apenas métodos de object

# Type safety e partilha de código em run-time

## Genéricos C# / Java

↳ Podem ser verificados antes que qualquer construção type seja construído para um conjunto de argumentos de tipo

↳ Ex: usando cláusulas where

⇒ a verificação é então válida para todos os argumentos de tipo possíveis

⇒ os erros são descobertos o mais cedo possível

## C++

Uma classe template não pode ser verificada antes que seja instanciado um tipo concreto, e o type-check é realizado separadamente para cada conjunto de argumentos. Ex: Stack<int> Stack<String>

## C#

Pair<double, int>
Pair<int, int>
} layout diferente mas sem box
↳ origina 2 novos tipos

Pair<student, string>
Pair<string, string>
} para tipos referência: o mesmo layout e código nativo é partilhado (manipulação de referências)

Tipos referência