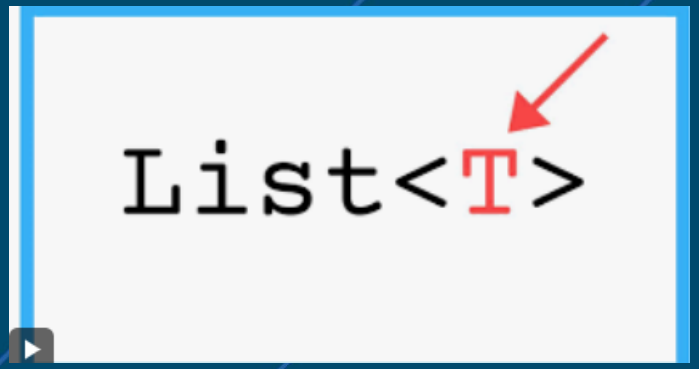
Féidearthachtaí as Cuimse
Infinite Possibilities



Generics in java

Object Oriented programming

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Scenario 1 run time error

```
List list = new ArrayList();
list.add("hello");
list.add(42); // This is allowed

String s = (String) list.get(1);
```

- No type checks → possible crash
- You must **cast manually (i.e. remember to manually change object types**

Scenario 2 compile time error

```
List<String> list = new ArrayList<>();
list.add("hello");

// list.add(42); // ❌ Compile-time error

String s = list.get(0); // ✅ No casting needed
```

- The compiler **enforces the type**
- Code is **cleaner** and **safer**

Generics

- Generics enable code that works **with any data type**.
- a **<placeholder>** for a data type (you'll see <T> or <E> usually)

- More flexible and reusable code
- Helps the compiler catch type errors (as opposed to runtime)
- No need for casting

Example 1: generics in Java collections

Map<K, V>: generic for key and value for creating a map

```
Map<String, Integer> studentScores = new
HashMap<>();
studentScores.put("Alice", 85);
studentScores.put("Bob", 92);

int score = studentScores.get("Alice"); //
```

Works the same way for List<T>, Set<T>, etc - specify object type T at coding time

Example 2: Generics in methods

- Supposing
- Printing different types of arrays..
- Multiple overloaded– see printArray() example
- Use a **generic method**

Example 2: Generics in methods

- Supposing a class that prints different types of arrays
- Multiple overloaded– see printArray() example
- Illogical !
- Use a **generic method**
  - **“whatever the type is, take it in at run time”**

recap

- Generics = **type-safe**, **reusable** code

- Use them with collections like List<T>, Map<K,V>

Generic methods let you write flexible utility code

- Most common generic types:
  T (Type), E (Element) (and K (Key), V (Value))
  But you can use any letter you want.. !!!