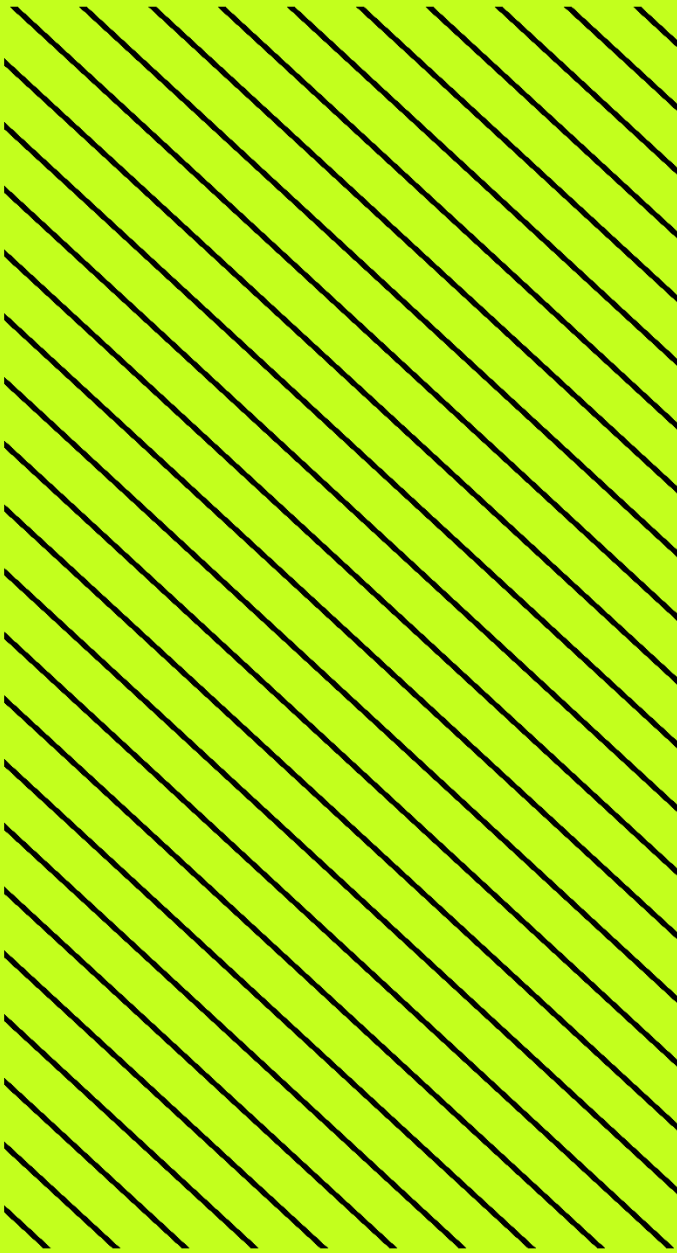
PROGRAMMING PARADIGMS

By Mohamed Baqer Ghazi



PROGRAMMING PARADIGMS

Templates in C++ vs Generics in Java

While building large-scale projects, we need the code to be compatible with any kind of data which is provided to it. That is the place where your written code stands above than that of others. Here what we meant is to make the code you write be generic to any kind of data provided to the program regardless of its data type. This is where Generics in Java and the similar in C++ named Template comes in handy. While both have similar functionalities, but they differ in a few places.

Template in C++:

Writing Generic programs in C++ is called Templates.
One of the major features of the template in C++ is the usage of metaprogramming. It Let the template signature of the provided code be different, were C++ provides the ability to implement them.
Template arguments can be both classes and in functions.
C++ requires template sources to be added in their headers.
Template specialization could be achieved i.e, Specific type and method of template can be implemented.

Example Code:

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
inline T const& Max (T const& a, T const& b) {
  return a < b ? b:a;
}
int main () {
  int i = 39;
  int j = 20;
  cout << "Max(i, j): " << Max(i, j) << endl;
  double f1 = 13.5;
  double f2 = 20.7;
  cout << "Max(f1, f2): " << Max(f1, f2) << endl;
  string s1 = "Hello";
  string s2 = "World";
  cout << "Max(s1, s2): " << Max(s1, s2) << endl;
  return 0; }
```

Output

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

PROGRAMMING PARADIGMS

Generics in Java:

One of the major features of Java Generics is that It handles type checking during instantiation and generates byte-code equivalent to non-generic code. The compiler of Java checks type before instantiation, that in turn makes the implementation of Generic type-safe. Meanwhile, in C++, templates know nothing about types. If Generics is applied in a class, then it gets Applied to classes and methods within classes. Another major factor that leads to the use of generics in Java is because it allows you to eliminate downcasts.
Instantiating a generic class has no runtime overhead over using an equivalent class that uses as specific *object* rather than a generic type of *T*

Example Code:

```
public class GenericMethodTest {
  // generic method printArray
  public static < E > void printArray( E[] inputArray ) {
    // Display array elements
    for(E element : inputArray) {
      System.out.printf("%s ", element);  }
    System.out.println();  }
  public static void main(String args[]) {
    // Create arrays of Integer, Double and Character
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
    System.out.println("Array integerArray contains:");
    printArray(intArray); // pass an Integer array
    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray); // pass a Double array
    System.out.println("\nArray characterArray contains:");
    printArray(charArray); // pass a Character array  } }
```

Output

Array integerArray contains:

1 2 3 4 5

Array doubleArray contains:

1.1 2.2 3.3 4.4

Array characterArray contains:

H E L L O

PROGRAMMING PARADIGMS

Comparing Templates and Generics:

Key differences between generics and C++ templates:

- Generics are generic until the types are substituted for them at runtime. Templates are specialized at compile time so they are not still parameterized types at runtime
- The common language runtime specifically supports generics in MSIL. Because the runtime knows about generics, specific types can be substituted for generic types when referencing an assembly containing a generic type. Templates, in contrast, resolve into ordinary types at compile time and the resulting types may not be specialized in other assemblies.
- Generics specialized in two different assemblies with the same type arguments are the same type. Templates specialized in two different assemblies with the same type arguments are considered by the runtime to be different types.
- Generics are generated as a single piece of executable code which is used for all reference type arguments (this is not true for value types, which have a unique implementation per value type). The JIT compiler knows about generics and is able to optimize the code for the reference or value types that are used as type arguments. Templates generate separate runtime code for each specialization.
- Generics do not allow non-type template parameters, such as template <int i> C {}. Templates allow them.
- Generics do not allow explicit specialization (that is, a custom implementation of a template for a specific type). Templates do.
- Generics do not allow partial specialization (a custom implementation for a subset of the type arguments). Templates do.
- Generics do not allow the type parameter to be used as the base class for the generic type. Templates do.
- Templates support template-template parameters (e.g. template<template<class T> class X> class MyClass), but generics do not.