23.11 The Implementation of Generic Types and Methods

Generic types and methods in C# resemble C++ type templates and function templates, and resemble generic types and methods in Java 1.5. However, the design and implementation of C# generics provide better type safety and less code duplication than C++ templates, and provides better performance and more features than Java 1.5 generic types and methods.

In C#, a generic type declaration or generic method declaration can be fully type-checked before any constructed type is created for given type arguments, and then the type-check is valid for all possible type arguments. Hence type errors are discovered as early as possible. In C++, a class template or function template cannot be type-checked before a type instance is created for given type parameters, and the type check must be performed separately for every set of given type parameters.

In C#, each constructed type, such as `Pair<double, int>`, where Pair is from example 182, in principle gives rise to a new type (in the run-time system) with its own field layout; this permits unboxed storage of fields. To save space and code duplication, every combination of type arguments that are reference types will share the same field layout: all references have the same size regardless of type. Thus the two constructed types `Pair<DateTime, String>` and `Pair<String, String>` have the same field layouts. On the other hand, different combinations of type arguments that are value types may require different field layouts at run-time, because values of different value types take up different amounts of memory. For instance, the constructed struct types `Pair<double, int>` and `Pair<int, int>` have different field layouts.

This implementation technique for C# has the considerable advantage that it permits generic types and generic methods to handle value types unboxed, which reduces memory usage and increases speed.

By contrast, the Java 1.5 implementation of generics uses boxed representation of all values of type parameters, which requires costly boxing, heap references, and unboxing when accessing the values.

On the other hand, the C# implementation has the disadvantage that it may cause some code duplication (at run-time), but this problem is much less pronounced than for C++ templates.

(*In*: "C# Precisely", Peter Sestoft and Henrik I. Hansen, The MIT Press, 2004.)