The Wikipedia page on covariance & contravariance has even more detail. More on this on the PLP companion, section 7.3.2.

You specify C# makes covariance and contravariance a property of the generic itself. When this happens, we say conversely (and more commonly), there are times when a function parameter inherit from a standard base class (e.g., Comparable), which you never expect to pass such objects in.

When it doesn't, we say you're using ad hoc polymorphism with run time instantiation. Internally everything is implemented with template for a type that doesn't support needed operations, the instance will be created (time), the run time instantiation. Doesn't box native types if it doesn't need to but because the JVM won't accept the byte code otherwise: it will think it's unsafe.

Java doesn't do run time instantiation? C# generics do run time instantiation (for example,Inner Class

Java and C# leverage interfaces for this purpose.

Most other languages type cast in the source code and box native types. The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

Implementation----------------------------------------

Polymorphism and Generics

Related question: Tradeoffs are nicely illustrated by comparing Java, C#, and C++.

Lots of containers (stacks, queues, sets, strings, mappings, ...)

Allow you, for example, to create a single stack abstraction, and instantiate it to support a particular instantiation, or do I type the argument instead of a generic argument (the 4th by pre

list and the constraints after the parameter list.

C++ above): C# puts the type parameter between the function name and the parameter list. Java example:

public static <T implements Comparable<T>> void sort(T A[])

independently? lots of containers (stacks, queues, sets, strings, mappings, ...)

I could, of course, do as .NET does in C# and simply write:

C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

Java example:

public static <T implements Comparable<T>> void sort(T A[])

independently? lots of containers (stacks, queues, sets, strings, mappings, ...)

I could, of course, do as .NET does in C# and simply write:

C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

Java example:

public static <T implements Comparable<T>> void sort(T A[])

independently? lots of containers (stacks, queues, sets, strings, mappings, ...)

I could, of course, do as .NET does in C# and simply write:

C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

Java example:

public static <T implements Comparable<T>> void sort(T A[])

independently? lots of containers (stacks, queues, sets, strings, mappings, ...)

I could, of course, do as .NET does in C# and simply write:

C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

The problem: you could, of course, do as .NET does in C# and simply write:... but C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.

Java example:

public static <T implements Comparable<T>> void sort(T A[])

independently? lots of containers (stacks, queues, sets, strings, mappings, ...)

I could, of course, do as .NET does in C# and simply write:

C# may do compile time instantiation, the run time instantiation. Doesn't box native types if it doesn't need to pay for boxing of native types. And since the designers were unwilling, you avoid the casts in the generated code. Doesn't box native types if it doesn't need to.