

Invariance, Co- and Contravariance

Say we have a small class hierarchy (here, in C++):

```
class A { ... };  
class B : public A { virtual B* m (B* x); };  
class C : public B  
{ < we plan a redefinition of m here > };
```

Now, what signature can be used for the redefinition?

Must it be identical to the signature of m in B ?

Or can we change the types for some parameters and the return value?

Motivation

What do you expect? Which signature could be meaningful?

- ▶ Certainly, using exactly the same signature as in class *B* is a very natural choice in many cases
- ▶ But consider a method **clone**: It should make a copy of the object and return that copy.
- ▶ In class *B*, clone should be: `virtual B* clone ();`
But in class *C*, we rather want to have `virtual C* clone();`
- ▶ So it would be useful if the language allowed such a modification of the signature

Invariance etc.

Allowed signature(s) for the redefinition depends on the language:

- ▶ Note: Invariance etc. is not a property of the signature as a whole, it can be different for, say, return type and types of the parameters
- ▶ **Invariance**: No change allowed (type must be the same)
- ▶ **Covariance**: Original type can be replaced with a subtype (subclass). This is what we wanted for the clone() method!
- ▶ **Contravariance** (very seldom): Original type can be replaced with some superclass

Examples

In C++

- ▶ The (implicit) *this* parameter (= the controlling argument) is always covariant
- ▶ Invariance for other parameters
- ▶ Invariance and covariance allowed for return type

In Java, it's basically the same, but till version 1.5, only invariance was allowed for return types.

In Ada, all controlling arguments (this can include the return type) are covariant, everything else is invariant.