Generally, the notation

=======================

[S <: T] means: **S is a subtype of T,(Upper bound).**

[S <: NonEmpty] S is all sub classes of NonEmpty ( BlackTree, RedTree etc)

[S >: T] means: **S is all supertype of T. (Lower bound)**

[S >: NonEmpty] could be one of NonEmpty, IntSet, AnyRef, or Any.

B :> A means that B is a super type of A and A is a sub type of B. Read righ to lelf . The operator next to the : is the super type.

Covariant

=========

If A <: B and C[A] <: C[B] -> ( A is subtype of B) and C[A] is also subtype of C[B]

**then C is covariant.**

If A <: B and C[A] >: C[B] -> ( A is subtype of B) and C[A] is also super of C[B]

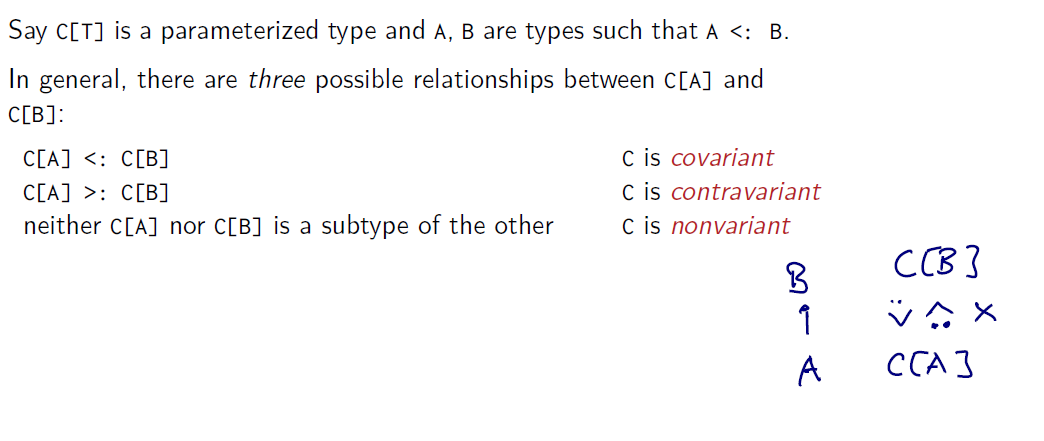
**then C is contra-covariant.**

\* **Type that cannot be covariant are:**  Types that accepts mutations of its elements should not be covariant. **For example Array.**

\***Type that can be covariant are:** immutable types can be covariant, if some conditions on the methods are met. **For example List (is immutable)** should be covariant whereas Array (is mutable) should be not.

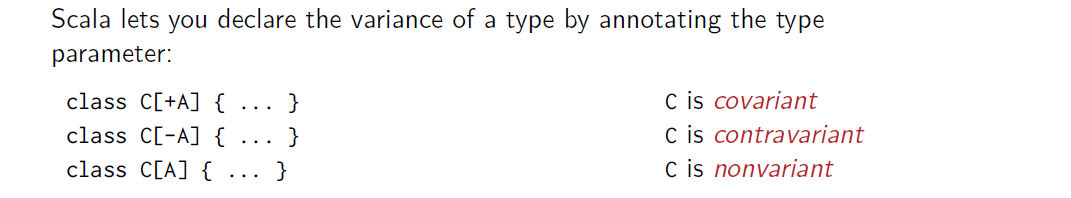
Possible relationship

=================

**Get/Put Principle**

In summary, we use covariance when we only intend to take generic values out of a structure. We use contravariance when we only intend to put generic values into a structure and we use an invariant when we intend to do both.

**Notation is scala**



\*) We can say that class C[+A] is always covariant or C[ -A] contra-varinat and etc.

**The Rule**

class C [+A] method[B>:A ] (parameter:B) {}

class C [-A] method[B<: A ] (parameter:B) {}

Nice rule

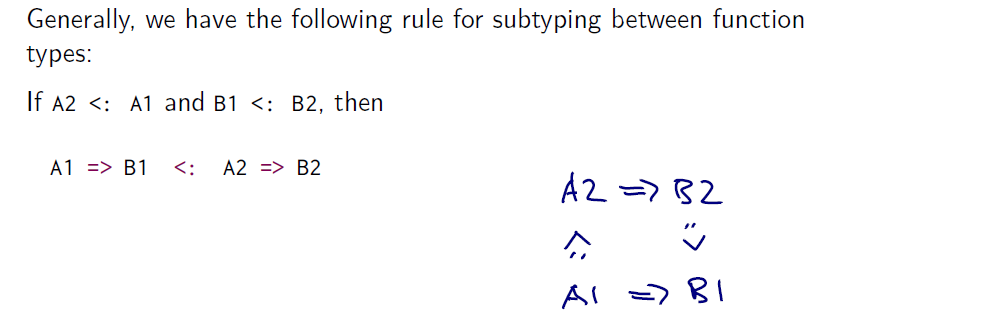
========

If A <: B, then everything one can to do with a value of type B one should also be able to do with a value of type A. For example all method in Object.

Nice rule for functions

====================

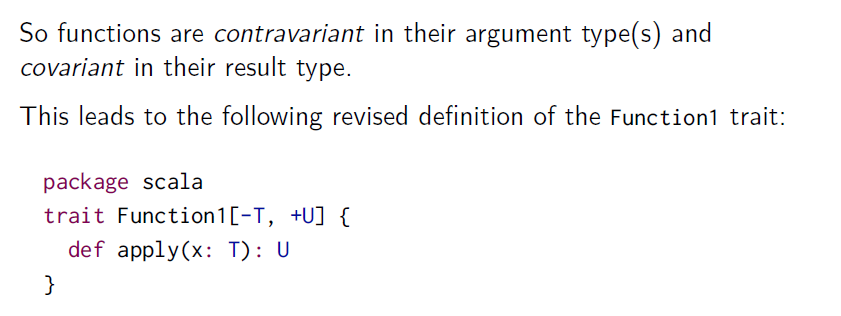
F1 is A1=>B1 is a method that takes A1 and return B1. F2 is A2=>B2 is a method that takes A2 and return B2. A2 is a subtype A1. B1 is a subtype of B2.



**\* F1 result <: F2 result**

Rule2 that compiler apply for +A and –A notation

===========================================



**\*This rule was created to prevent mutable actions is covariant classes.**

Scala compiler checks

==================

