TOUR OF SCALA

LOWER TYPE BOUNDS

While upper type bounds limit a type to a subtype of another type, *lower type bounds* declare a type to be a supertype of another type. The term `B >: A` expresses that the type parameter `B` or the abstract type `B` refer to a supertype of type `A`. In most cases, `A` will be the type parameter of the class and `B` will be the type parameter of a method.

Here is an example where this is useful:

```
trait Node[+B] {
  def prepend(elem: B): Node[B]
}

case class ListNode[+B](h: B, t: Node[B]) extends Node[B] {
  def prepend(elem: B): ListNode[B] = ListNode(elem, this)
  def head: B = h
  def tail: Node[B] = t
}

case class Nil[+B]() extends Node[B] {
  def prepend(elem: B): ListNode[B] = ListNode(elem, this)
}
```

This program implements a singly-linked list. `Nil` represents an empty element (i.e. an empty list). `class ListNode` is a node which contains an element of type `B` (`head`) and a reference to the rest of the list (`tail`). The `class Node` and its subtypes are covariant because we have `+B`.

However, this program does *not* compile because the parameter `elem` in `prepend` is of type `B`, which we declared *co*variant. This doesn't work because functions are *contra*variant in their parameter types and *co*variant in their result types.

To fix this, we need to flip the variance of the type of the parameter `elem` in `prepend`. We do this by introducing a new type parameter `U` that has `B` as a lower type bound.

```
trait Node[+B] {
  def prepend[U >: B](elem: U): Node[U]
}

case class ListNode[+B](h: B, t: Node[B]) extends Node[B] {
  def prepend[U >: B](elem: U): ListNode[U] = ListNode(elem, this)
  def head: B = h
  def tail: Node[B] = t
}

case class Nil[+B]() extends Node[B] {
  def prepend[U >: B](elem: U): ListNode[U] = ListNode(elem, this)
}
```

Now we can do the following:

```
trait Bird
case class AfricanSwallow() extends Bird
case class EuropeanSwallow() extends Bird



val africanSwallowList = ListNode[AfricanSwallow](AfricanSwallow(), Nil())
val birdList: Node[Bird] = africanSwallowList
birdList.prepend(EuropeanSwallow())
```

The `Node[Bird]` can be assigned the `africanSwallowList` but then accept `EuropeanSwallow`s.

← previous next →

Contributors to this page:

ckipp01 mlachkar vegerot Mõškovski Stanislav ashawley ipostanogov justinpermar jvican lierdakil heathermiller Andriy Yurchuk