TOUR OF SCALA

ABSTRACT TYPE MEMBERS

Abstract types, such as traits and abstract classes, can in turn have abstract type members. This means that the concrete implementations define the actual types. Here's an example:

```
trait Buffer {
  type T
  val element: T
}
```

Here we have defined an abstract `type T`. It is used to describe the type of `element`. We can extend this trait in an abstract class, adding an upper-type-bound to `T` to make it more specific.

```
abstract class SeqBuffer extends Buffer {
  type U
  type T <: Seq[U]
  def length = element.length
}
```

Notice how we can use yet another abstract type `U` in the specification of an upper-type-bound for `T`. This `class SeqBuffer` allows us to store only sequences in the buffer by stating that type `T` has to be a subtype of `Seq[U]` for a new abstract type `U`.

Traits or classes with abstract type members are often used in combination with anonymous class instantiations. To illustrate this, we now look at a program which deals with a sequence buffer that refers to a list of integers:

```
abstract class IntSeqBuffer extends SeqBuffer {
  type U = Int
}

def newIntSeqBuf(elem1: Int, elem2: Int): IntSeqBuffer =
  new IntSeqBuffer {
       type T = List[U]
       val element = List(elem1, elem2)
     }
val buf = newIntSeqBuf(7, 8)
println("length = " + buf.length)
println("content = " + buf.element)
```

Here the factory `newIntSeqBuf` uses an anonymous class implementation of `IntSeqBuffer` (i.e. `new IntSeqBuffer`) to set the abstract type `T` to the concrete type `List[Int]`.

It is also possible to turn abstract type members into type parameters of classes and vice versa. Here is a version of the code above which only uses type parameters:

```
abstract class Buffer[+T] {
  val element: T

}
abstract class SeqBuffer[U, +T <: Seq[U]] extends Buffer[T] {
  def length = element.length
```

```
  def length = element.length
}

def newIntSeqBuf(e1: Int, e2: Int): SeqBuffer[Int, Seq[Int]] =
  new SeqBuffer[Int, List[Int]] {
    val element = List(e1, e2)
  }

val buf = newIntSeqBuf(7, 8)
println("length = " + buf.length)
println("content = " + buf.element)
```

Note that we have to use variance annotations here (`+T <: Seq[U]`) in order to hide the concrete sequence implementation type of the object returned from method `newIntSeqBuf` . Furthermore, there are cases where it is not possible to replace abstract type members with type parameters.

← previous next →

Contributors to this page: