Dropped: Auto-Application

Edit this page on GitHub

Previously an empty argument list `()` was implicitly inserted when calling a nullary method without arguments. Example:

```
def next(): T = ...
next     // is expanded to next()
```

In Scala 3, this idiom is an error.

```
next
^
missing arguments for method next
```

In Scala 3, the application syntax has to follow exactly the parameter syntax. Excluded from this rule are methods that are defined in Java or that override methods defined in Java. The reason for being more lenient with such methods is that otherwise everyone would have to write

```
xs.toString().length()
```

instead of

```
xs.toString.length
```

The latter is idiomatic Scala because it conforms to the *uniform access principle*. This principle states that one should be able to change an object member from a field to a non-side-effecting method and back without affecting clients that access the member. Consequently, Scala encourages to define such "property" methods without a `()` parameter list whereas side-effecting methods should be defined with it. Methods defined in Java cannot make this distinction; for them a `()` is always mandatory. So Scala fixes the problem on the client side, by allowing the parameterless references. But where Scala allows that freedom for all method

references, Scala 3 restricts it to references of external methods that are not defined themselves in Scala 3.

For reasons of backwards compatibility, Scala 3 for the moment also auto-inserts `()` for nullary methods that are defined in Scala 2, or that override a method defined in Scala 2. It turns out that, because the correspondence between definition and call was not enforced in Scala so far, there are quite a few method definitions in Scala 2 libraries that use `()` in an inconsistent way. For instance, we find in `scala.math.Numeric`

```
def toInt(): Int
```

whereas `toInt` is written without parameters everywhere else. Enforcing strict parameter correspondence for references to such methods would project the inconsistencies to client code, which is undesirable. So Scala 3 opts for more leniency when type-checking references to such methods until most core libraries in Scala 2 have been cleaned up.

Stricter conformance rules also apply to overriding of nullary methods. It is no longer allowed to override a parameterless method by a nullary method or *vice versa*. Instead, both methods must agree exactly in their parameter lists.

```
class A:
  def next(): Int

class B extends A:
  def next: Int // overriding error: incompatible type
```

Methods overriding Java or Scala 2 methods are again exempted from this requirement.

Migrating code

Existing Scala code with inconsistent parameters can still be compiled in Scala 3 under `-source 3.0-migration`. When paired with the `-rewrite` option, the code will be automatically rewritten to conform to Scala 3's stricter checking.

Reference

For more information, see Issue #2570 and PR #2716.

Dropp... Dropp...