Universal Apply Methods

Edit this page on GitHub

Scala case classes generate apply methods, so that values of case classes can be created using simple function application, without needing to write `new`.

Scala 3 generalizes this scheme to all concrete classes. Example:

```
class StringBuilder(s: String):
  def this() = this("")

StringBuilder("abc")  // old: new StringBuilder("abc")
StringBuilder()       // old: new StringBuilder()
```

This works since a companion object with two `apply` methods is generated together with the class. The object looks like this:

```
object StringBuilder:
  inline def apply(s: String): StringBuilder = new StringBuilder(s)
  inline def apply(): StringBuilder = new StringBuilder()
```

The synthetic object `StringBuilder` and its `apply` methods are called *constructor proxies*. Constructor proxies are generated even for Java classes and classes coming from Scala 2. The precise rules are as follows:

1. A constructor proxy companion object `object C` is created for a concrete class `C`, provided the class does not have already a companion, and there is also no other value or method named `C` defined or inherited in the scope where `C` is defined.

2. Constructor proxy `apply` methods are generated for a concrete class provided

    - the class has a companion object (which might have been generated in step 1), and
    - that companion object does not already define a member named `apply`

- that companion object does not already define a member named `apply`.

  Each generated `apply` method forwards to one constructor of the class. It has the same type and value parameters as the constructor.

Constructor proxy companions cannot be used as values by themselves. A proxy companion object must be selected with `apply` (or be applied to arguments, in which case the `apply` is implicitly inserted).

Constructor proxies are also not allowed to shadow normal definitions. That is, if an identifier resolves to a constructor proxy, and the same identifier is also defined or imported in some other scope, an ambiguity is reported.

Motivation

Leaving out `new` hides an implementation detail and makes code more pleasant to read. Even though it requires a new rule, it will likely increase the perceived regularity of the language, since case classes already provide function call creation syntax (and are often defined for this reason alone).

‹ Transp… Export … ›

Scaladoc Copyright (c) 2002-2022, LAMP/EPFL