Importing Givens

Edit this page on GitHub

A special form of import wildcard selector is used to import given instances. Example:

```
object A:
  class TC
  given tc: TC = ???
  def f(using TC) = ???

object B:
  import A.*
  import A.given
  ...
```

In the code above, the `import A.*` clause in object `B` imports all members of `A` *except* the given instance `tc`. Conversely, the second import `import A.given` will import *only* that given instance. The two import clauses can also be merged into one:

```
object B:
  import A.{given, *}
   ...
```

Generally, a normal wildcard selector `*` brings all definitions other than givens or extensions into scope whereas a `given` selector brings all givens (including those resulting from extensions) into scope.

There are two main benefits arising from these rules:

- It is made clearer where givens in scope are coming from. In particular, it is not possible to hide imported givens in a long list of regular wildcard imports.
- It enables importing all givens without importing anything else. This is particularly important since givens can be anonymous, so the usual recourse of using named imports is not practical.

Importing By Type

Since givens can be anonymous it is not always practical to import them by their name, and wildcard imports are typically used instead. By-type imports provide a more specific alternative to wildcard imports, which makes it clearer what is imported. Example:

```
import A.given TC
```

This imports any given in `A` that has a type which conforms to `TC`. Importing givens of several types `T1, ... ,Tn` is expressed by multiple `given` selectors.

```
import A.{given T1, ..., given Tn}
```

Importing all given instances of a parameterized type is expressed by wildcard arguments. For instance, assuming the object

```
object Instances:
  given intOrd: Ordering[Int] = ...
  given listOrd[T: Ordering]: Ordering[List[T]] = ...
  given ec: ExecutionContext = ...
  given im: Monoid[Int] = ...
```

the import clause

```
import Instances.{given Ordering[?], given ExecutionContext}
```

would import the `intOrd`, `listOrd`, and `ec` instances but leave out the `im` instance, since it fits none of the specified bounds.

By-type imports can be mixed with by-name imports. If both are present in an import clause, by-type imports come last. For instance, the import clause

```
import Instances.{im, given Ordering[?]}
```

would import `im`, `intOrd`, and `listOrd` but leave out `ec`.

Migration

The rules for imports stated above have the consequence that a library would have to migrate in lockstep with all its users from old style implicits and normal imports to givens and given imports.

givens and given imports.

The following modifications avoid this hurdle to migration.

1. A `given` import selector also brings old style implicits into scope. So, in Scala 3.0 an old-style implicit definition can be brought into scope either by a `*` or a `given` wildcard selector.
2. In Scala 3.1, old-style implicits accessed through a `*` wildcard import will give a deprecation warning.
3. In some version after 3.1, old-style implicits accessed through a `*` wildcard import will give a compiler error.

These rules mean that library users can use `given` selectors to access old-style implicits in Scala 3.0, and will be gently nudged and then forced to do so in later versions. Libraries can then switch to given instances once their user base has migrated.

Syntax

```
Import            ::=  ‘import’ ImportExpr {‘,’ ImportExpr}
Export            ::=  ‘export’ ImportExpr {‘,’ ImportExpr}
ImportExpr        ::=  SimpleRef {‘.’ id} ‘.’ ImportSpec
ImportSpec        ::=  NamedSelector
                    |  WildcardSelector
                    | ‘{’ ImportSelectors) ‘}’
NamedSelector     ::=  id [‘as’ (id | ‘_’)]
WildCardSelector  ::=  ‘*' | ‘given’ [InfixType]
ImportSelectors   ::=  NamedSelector [‘,’ ImportSelectors]
                    |  WildCardSelector {‘,’ WildCardSelector}
```

‹ Contex... Extensi... ›

Contributors to this page

pikinier20 BarkingBad julienrf odersky jrudolph
michelou ShapelessCat

Scaladoc