Dropped: Package Objects

Package objects

```
package object p {
  val a = ...
  def b = ...
}
```

will be dropped. They are still available in Scala 3.0 and 3.1, but will be deprecated and removed afterwards.

Package objects are no longer needed since all kinds of definitions can now be written at the top-level. Example:

```
package p
type Labelled[T] = (String, T)
val a: Labelled[Int] = ("count", 1)
def b = a._2

case class C()

extension (x: C) def pair(y: C) = (x, y)
```

There may be several source files in a package containing such top-level definitions, and source files can freely mix top-level value, method, and type definitions with classes and objects.

The compiler generates synthetic objects that wrap top-level definitions falling into one of the following categories:

- all pattern, value, method, and type definitions,
- implicit classes and objects,
- companion objects of opaque type aliases.

If a source file `src.scala` contains such top-level definitions, they will be put in a synthetic object named `src$package`. The wrapping is transparent, however. The definitions in `src` can still be accessed as members of the enclosing package. The synthetic object will be placed last in the file, after any other package clauses, imports, or object and class definitions.

Note: This means that

1. The name of a source file containing wrapped top-level definitions is relevant for binary compatibility. If the name changes, so does the name of the generated object and its class.
2. A top-level main method `def main(args: Array[String]): Unit = ...` is wrapped as any other method. If it appears in a source file `src.scala`, it could be invoked from the command line using a command like `scala src$package`. Since the "program name" is mangled it is recommended to always put `main` methods in explicitly named objects.
3. The notion of `private` is independent of whether a definition is wrapped or not. A `private` top-level definition is always visible from everywhere in the enclosing package.
4. If several top-level definitions are overloaded variants with the same name, they must all come from the same source file.

‹ Dropp... Dropp... ›