Packages and imports

Any Kotlin project is structured into **packages**. A package may contain one or more Kotlin files and each file is related to the corresponding package using the *package header*. A file may contain only one (or zero) package headers, meaning that each file belongs to exactly one package.

Id grammar-rule-packageHeader not found

> Note: an absence of a package header in a file means that is belongs to the special *root package*
>
> Note: Packages are different from modules. A module may contain many packages, while a single package can be spread across several modules.

The name of a package is a dot (`.`)-separated sequence of identifiers, introducing a package hierarchy. Unlike Java and some other languages, Kotlin does not restrict the package hierarchy to correspond directly to the folder structure of the project.

> Note: this means that the hierarchy itself is only notational, not affecting the code in any way. It is strongly recommended, however, that the folder structure of the project does correspond to the package hierarchy.

Importing

Program entities declared in one package may be freely used in any file in the same package with the only restriction being module boundaries. In order to use an entity from a file belonging to a different package, the programmer must use *import directives*.

Id grammar-rule-importList not found

Id grammar-rule-importHeader not found

Id grammar-rule-importAlias not found

An import directive contains dot-separated *path* to an entity, as well as the name of the entity itself (the last argument of the navigation dot operator). A path may include not only the package the import is importing from, but also an object or a type (referring to companion object of this type). Any named declaration within that scope (that is, top-level scope of all files in the package or, in the object case, the object declararion scope) may be imported using their names. There are two special kinds of imports: star-imports ending in an asterisk (`*`) and renaming imports employing the use of `as` operator. Star-imports import all the named entities inside the corresponding scope, but have weaker priority during [resolution][Overload resolution] of functions and properties. Renaming

imports work just as regular imports, but introduce the entity into current file with a name different from the name it has at declaration site.

Imports are file-based, meaning that if an entity is introduced into file A.kt belonging to package `kotlinx.foo`, it does not introduce this entity to all other files belonging to `kotlinx.foo`.

There are some packages that have all their entities *implicitly imported* into any Kotlin file, meaning one can access this entity without explicitly using import directives. One may, however, import this entities explicitly if they choose to. These are the following packages of the standard library:

- `kotlin`
- `kotlin.annotation`
- `kotlin.collections`
- `kotlin.comparisons`
- `kotlin.io`
- `kotlin.ranges`
- `kotlin.sequences`
- `kotlin.text`
- `kotlin.math`

Platform implementations may introduce additional implicitly imported packages, for example, adding standard platform functionality into Kotlin code.

> Note: an example of this would be `java.lang` package implicitly imported on the jvm platform

Importing certain entities may be disallowed by their [visibility modifiers][Visibility].

TODO(Clarify all this)

Modules

TODO(Here be The dragons)