Scopes and identifiers

All the program code in Kotlin is logically divided into *scopes*. A scope is a syntactically-delimited region of code that constitutes a context in which entities and their names can be introduced. Scopes are nested, with entities introduced in outer scopes also available in the inner scopes. The top level of a Kotlin file is also a scope, containing all the scopes within the file.

All the scopes are divided into two categories: declaration scopes and statement scopes. These two kinds of scopes differ in how the identifiers in code refer to the values definied in the scopes.

Declaration scopes include:

- The top level scope of a normal Kotlin file (not script file);
- The bodies of [classifier declarations][Classifier declaration];
- The bodies of [object literals][Object literals];
- TODO(Anything else?)

Statement scopes include:

- The top level scope of a Kotlin script file;
- Various scopes produced by control structure bodies of different [expressions][Expressions];
- The bodies of [function declarations][Function declaration];
- The bodies of [anonymous function literals][Anonymous function declarations];
- The bodies of getters and setters of [properties][Property declaration];
- The bodies of [constructors][Constructor declaration];
- The bodies of instance initialization blocks in [class declarations][Class declaration];
- TODO(Anything else?)

All the declarations in a particular scope introduce new *bindings* of identifiers in this scope to their respective entities in the program. These entities may be types or values, where values may refer to objects, functions or properties (that may be delegated). Top-level scopes additionally allow to introduce such bindings using [`import` directive][Packages and imports] from other top-level scopes.

(TODO: what's a signature?)

In most situations, it is not allowed to bind several values to the same identifier in the same scope, but it is allowed to bind a value to an identifier already available in the scope through outer scopes or imports. An exception to this rule are function declarations, that, in addition to identifier bound to, also may differ by signature and allow definining several functions with the same name in the same scope. When [calling functions][Call and property access expressions] a process called [overloading resolution][Overload resolution] takes places that allows differentiating such functions. Overloading resolution also applies to properties if they are used as functions through `invoke`-convention, but it does not mean several properties with the same name may be defined in the same scope.

The main difference between declaration scopes and statement scopes is that names in the statement scope are bound in the order their declarations appear in it. It is not allowed to access a value through an identifier in the code that (syntactically) precedes the binding itself. On the contrary, in declaration scopes it is fully allowed, although initialization cycles may occur and need to be detected by the compiler. It also means that the statement scopes nested inside declaration scopes may access values declared after itself in the declaration scopes, but any values defined inside the statement scope must be accessed only after they are declared.

Example:

- In declaration scope:

  ```
  // x refers to the property defined below even if there is another property
  // called x in outer scope or imported
  fun foo() { return x + 2; }
  val x = 3;
  ```

- In statement scope:

  ```
  // x either refers to other property defined in some outer scope or imported
  // or it is a compile-time error
  fun foo() { return x + 2; }
  val x = 3;
  ```

  Note: please note that all the above is primarily applied to declarations, because declaration scopes do not allow standalone statements to appear in them

- TODO(qualified names?)
- TODO(extensions?)
- TODO(receivers)

- TODO(rewrite expressions and statements as references to this part)
- TODO(identifier lifetime & such)