Kotlin language specification

Version 1.9-rfc+0.1

Marat Akhin Mikhail Belyaev

Chapter 3

Built-in types and their semantics

Kotlin has several built-in classifier types, which are important for the rest of this document. Some information about these types is given in the type system section, here we extend it with additional non-type-system-relevant details.

> Note: these types may have regular declarations in the standard library, but they also introduce semantics not representable via Kotlin source code.

In this section we describe these types and their semantics.

> Note: this section is not meant to be a detailed description of all types available in the standard library, for that please refer to the standard library documentation.

3.1 `kotlin.Any`

Besides being the unified supertype of all non-nullable types, `kotlin.Any` must also provide the following methods.

-
  ```
  public open operator fun equals(other: Any?): Boolean
  ```

  Returns `true` iff a value is equal to some other value. Implementations of `equals` must satisfy the properties of reflexivity (`x.equals(x)` is always true), symmetry (`x.equals(y) == y.equals(x)`), transitivity (if `x.equals(y)` is true and `y.equals(z)` is true, `x.equals(z)` is also true) and consistency (`x.equals(y)` should not change its result between multiple invocations). A non-null value also must never be considered equal to `null`, i.e., `x.equals(null)` must be `false`.

- `public open fun hashCode(): Int`

  Returns the hash code for a value. Implementations of `hashCode` must satisfy the following property: if two values are equals w.r.t. `equals`, `hashCode` must consistently produce the same result.

- `public open fun toString(): String`

  Returns a string representation of a value.

3.2 `kotlin.Nothing`

`kotlin.Nothing` is an uninhabited type, which means the evaluation of an expression with `kotlin.Nothing` type can never complete normally. Therefore, it is used to mark special situations, such as

- non-terminating expressions
- exceptional control flow
- control flow transfer

Further details about how `kotlin.Nothing` should be handled are available here and here.

3.3 `kotlin.Unit`

`kotlin.Unit` is a unit type, i.e., a type with only one value `kotlin.Unit`; all values of type `kotlin.Unit` should reference the same underlying `kotlin.Unit` object. It is somewhat similar in purpose to `void` return type in other programming languages in that it signifies an *absence of a value* (i.e. the returned type for a function returning nothing), but is different in that there is, in fact, a single value of this type.

3.4 `kotlin.Boolean`

`kotlin.Boolean` is the boolean logic type of Kotlin, representing a value which may be either `true` or `false`. It is the type of boolean literals as well as the type returned or expected by some built-in Kotlin operators.

3.5 Built-in integer types

Kotlin has several built-in classifier types, which represent signed integer numbers of different bit size. These types are important w.r.t. type system and integer literals. Every built-in integer type `I` is a subtype of `kotlin.Comparable<I>`.

The signed integer types are the following.

- `kotlin.Int`
- `kotlin.Short`
- `kotlin.Byte`
- `kotlin.Long`

> Note: Kotlin does not have a built-in arbitrary-precision integer type.

> Note: Kotlin does not have any built-in unsigned integer types.

These types may or may not have different runtime representations, depending on the used platform and/or implementation. Consult the specific platform reference for further details.

`kotlin.Int` is the type of integer numbers that is required to be able to hold the values at least in the range from -2^{31} to $2^{31}-1$. If an arithmetic operation on `kotlin.Int` results in arithmetic overflow, the result is unspecified.

`kotlin.Short` is the type of integer numbers that is required to be able to hold the values at least in the range from -2^{15} to $2^{15}-1$. If an arithmetic operation on `kotlin.Short` results in arithmetic overflow, the result is unspecified.

`kotlin.Byte` is the type of integer numbers that is required to be able to hold the values at least in the range from -2^{7} to $2^{7}-1$. If an arithmetic operation on `kotlin.Byte` results in arithmetic overflow, the result is unspecified.

`kotlin.Long` is the type of integer numbers that is required to be able to hold the values at least in the range from -2^{63} to $2^{63}-1$. If an arithmetic operation on `kotlin.Long` results in arithmetic overflow, the result is unspecified.

> Note: by “arithmetic overflow” we assume both positive and negative integer overflows.

> Important: a platform implementation may specify behaviour for an arithmetic overflow.

3.5.1 Integer type widening

In overload resolution, we actually have a priority between built-in integer types which is very similar to a subtyping relation between these types; however, this priority is important only w.r.t. overload resolution and does not entail any actual subtyping between built-in integer types.

In order to introduce this priority we describe a type transformation called *widening* of integer types. $\text{Widen}(T)$ for a built-in integer type T is defined as follows:

- $\text{Widen}(\texttt{kotlin.Int}) = \texttt{kotlin.Int} \,\&\, \texttt{kotlin.Short} \,\&\, \texttt{kotlin.Byte} \,\&\, \texttt{kotlin.Long}$
- $\text{Widen}(\texttt{kotlin.Short}) = \texttt{kotlin.Short} \,\&\, \texttt{kotlin.Byte}$
- $\text{Widen}(T) = T$ for any other T

> Informally: Widen means, for the purposes of overload resolution, `kotlin.Int` is preferred over any other built-in integer type and

kotlin.Short is preferred to kotlin.Byte. Using Widen, we can reduce this priority to subtyping: T is more preferred than U if $\text{Widen}(T) <: \text{Widen}(U)$; this scheme allows to handle built-in integer types transparently when selecting the most specific overload candidate.

For example, consider the following two functions:

```
fun foo(value: Int) = 1
fun foo(value: Short) = 2

...

foo(2)
```

As the integer literal 2 has a type that is applicable for both versions of foo (see Overload resolution section for details) and the types kotlin.Int and kotlin.Short are not related w.r.t. subtyping, it would not be possible to select a more specific candidate out of the two. However, if we consider Widen(kotlin.Int) and Widen(kotlin.Short) respectively as the types of value, first candidate becomes more specific than the second, because Widen(kotlin.Int) <: Widen(kotlin.Short).

3.6 Built-in floating point arithmetic types

There are two built-in classifier types which represent floating-point numbers: kotlin.Float and kotlin.Double. These types may or may not have different runtime representations, depending on the used platform and/or implementation. Consult the specific platform reference for further details.

kotlin.Float is the type of floating-point number that is able to contain all the numbers as a IEEE 754 single-precision binary floating number with the same precision. kotlin.Float is a subtype of kotlin.Comparable<kotlin.Float>.

kotlin.Double is the type of floating-point number that is able to contain all the numbers as a IEEE 754 double-precision binary floating number with the same precision. kotlin.Double is a subtype of kotlin.Comparable<kotlin.Double>.

Platform implementations may give additional information on how these types are represented on a particular platform.

3.7 kotlin.Char

kotlin.Char is the built-in classifier type which represents a single Unicode symbol in UCS-2 character encoding. It is the type of character literals.

> Important: a platform implementation may *extend* the supported character encodings, e.g., to UTF-16.

3.8 `kotlin.String`

`kotlin.String` is the built-in classifier type which represents a sequence of Unicode symbol in UCS-2 character encoding. It is the type of the result of string interpolation.

> Important: a platform implementation may *extend* the supported character encodings, e.g., to UTF-16.

3.9 `kotlin.Enum`

`kotlin.Enum<T>` is the built-in parameterized classifier type which is used as a superclass for all enum classes: every enum class `E` is implicitly a subtype of `kotlin.Enum<E>`.

`kotlin.Enum<T>` has the following characteristics.

- `kotlin.Enum<T>` is a subtype of `kotlin.Comparable<T>`

`kotlin.Enum<T>` provides the following properties.

-
  ```
  public final val name: String
  ```

  Contains the name of this enumeration constant, exactly as declared in its declaration.

-
  ```
  public final val ordinal: Int
  ```

  Contains the ordinal of this enumeration constant, i.e., its position in the declaration, starting from zero.

`kotlin.Enum<T>` provides the following member functions.

-
  ```
  public override final fun compareTo(other: T): Int
  ```

  The implementation of `kotlin.Comparable`. The result of `a.compareTo(b)` for enum class instances `a` and `b` is equivalent to `a.ordinal.compareTo(b.ordinal)`.

-
  ```
  public override final fun equals(other: Any?): Boolean
  ```

-
  ```
  public override final fun hashCode(): Int
  ```

  These member functions are defined to their default behaviour: only the same entry of an enum class is equal to itself and no other object. Hash implementation is required to be consistent, but unspecified.

  > Note: the presence of these final member functions ensures the semantics of equality and comparison for the enumeration objects, as they cannot be overridden by the user.

- `protected final fun clone(): Any`

  Throws an unspecified exception.

  > Note: the `clone()` implementation throws an exception, as enumeration objects cannot be copied and on some platforms `clone` function serves for copying.

3.10 Built-in array types

`kotlin.Array<T>` is the built-in parameterized classifier type which is used to represent an indexed fixed-size collection of elements of type `T`.

It is final (i.e., cannot be inherited from) and has the following public constructor.

- `public inline constructor(size: Int, init: (Int) -> T)`

  Creates a new array with the specified size, where each element is calculated by calling the specified `init` function with the corresponding element's index. The function `init` is called for each array element sequentially starting from the first one. This constructor is special in two ways: first, it is `inline` and inline constructors are not generally allowed in Kotlin. Second, it is required for the parameter `T` to be instantiated with a runtime-available type.

`kotlin.Array<T>` provides the following methods and properties.

- `public operator fun get(index: Int): T`

  Returns the array element at the specified index. If the [index] is out of bounds of this array, throws an `IndexOutOfBoundsException`.

- `public operator fun set(index: Int, value: T): Unit`

  Sets the array element at the specified index to the specified value. If the [index] is out of bounds of this array, throws an `IndexOutOfBoundsException`.

- `public val size: Int`

  Returns the array size.

- `public operator fun iterator(): Iterator<T>`

  Creates an iterator for iterating over the elements of the array.

3.10.1 Specialized array types

In addition to the general `kotlin.Array<T>` type, Kotlin also has the following specialized array types:

- `kotlin.DoubleArray` (for `kotlin.Array<kotlin.Double>`)

- `kotlin.FloatArray` (for `kotlin.Array<kotlin.Float>`)
- `kotlin.LongArray` (for `kotlin.Array<kotlin.Long>`)
- `kotlin.IntArray` (for `kotlin.Array<kotlin.Int>`)
- `kotlin.ShortArray` (for `kotlin.Array<kotlin.Short>`)
- `kotlin.ByteArray` (for `kotlin.Array<kotlin.Byte>`)
- `kotlin.CharArray` (for `kotlin.Array<kotlin.Char>`)
- `kotlin.BooleanArray` (for `kotlin.Array<kotlin.Boolean>`)

These array types are similar to the corresponding `kotlin.Array<T>` type; i.e., `kotlin.IntArray` has the same methods and properties as `kotlin.Array<Int>`, with the following changes.

- `public constructor(size: Int)`

  Creates a new array with the specified size, where each element is set to the corresponding built-in type default value.

  > Note: default values are platform-specific.

- `public operator fun iterator(): {TYPE}Iterator`

  Creates a specialized iterator for iterating over the elements of the array.

3.11 Iterator types

`kotlin.Iterator<out T>` is the built-in parameterized classifier type which is used to represent a sequence of elements of type `T`, allowing for sequential access to these elements.

It provides the following methods.

- `public operator fun next(): T`

  Returns the next element in the sequence.

- `public operator fun hasNext(): Boolean`

  Returns `true` if the sequence has more elements.

3.11.1 Specialized iterator types

Specialized iterator types are iterator types used for specialized array types. They inherit `kotlin.Iterator<out T>` for their type (i.e., `kotlin.CharIterator` inherits `kotlin.Iterator<Char>`) and provide the following methods.

- `public operator fun next{TYPE}(): {TYPE}`

  Returns the next element in the sequence as a specific type.

  > Note: this additional method allows the compiler and/or developer to avoid unneeded platform-specific boxing/unboxing conversions.

3.12 kotlin.Throwable

`kotlin.Throwable` is the built-in classifier type that is the base type of all exception types. Any value that is used in a `throw` expression must have a static type that is a subtype of `kotlin.Throwable`. Any type that is used in a `catch` part of the `try` expression must be a subtype of (or equal to) `kotlin.Throwable`.

It provides at least the following properties:

- `public val message: String?`

  An optional message depicting the cause of the throw.

- `public val cause: Throwable?`

  An optional other value of type `kotlin.Throwable` allowing for nested throwables to be constructed.

Other members may exist, please refer to the standard library documentation for details. No subtype of `kotlin.Throwable` is allowed to have type parameters. Declaring such a type is a compile-time error.

3.13 kotlin.Comparable

`kotlin.Comparable<in T>` is a built-in parameterized type which represents values that may be compared for total ordering. It provides the following member function:

```
public operator fun compareTo(other: T): Int
```

This function is used to implement comparison operators through overloadable operator convention for standard library classes.

> Note: a type is not required to be a subtype of `kotlin.Comparable` in order to implement total ordering operations

3.14 kotlin.Function

`kotlin.Function<out R>` is the base classifier type of all function types. See the relevant section for details.

3.15 Built-in annotation types

Kotlin has a number of built-in annotation types, which are covered in more detail here.

3.16 Reflection support builtin types

3.16.1 `kotlin.reflect.KClass`

`kotlin.reflect.KClass<T: Any>` is the class used to represent runtime type information for runtime-available classifier types. It is also used in platform-specific reflection facilities.

This is the type of class literals. This type is required to introduce `equals` and `hashCode` member function implementations (see `kotlin.Any`) that allow for comparison and hashing of runtime type information, e.g., that class literals are equal if they represent the same runtime type and not equal otherwise. Platform definitions, as well as particular implementations, may introduce additional members for this type.

3.16.2 `kotlin.reflect.KCallable`

`kotlin.reflect.KCallable<out R>` is the class used to represent runtime information for callables (i.e. properties and functions). It is mainly used as base type for other types described in this section. It provides at least the following property:

```
public val name: String
```

This property contains the name of the callable. Other members or base types for this class may be provided by platform and/or implementation.

3.16.3 `kotlin.reflect.KProperty`

`kotlin.reflect.KProperty<out R>` is the class used to represent runtime information for properties. It is the base type of property references. This type is used in property delegation. `kotlin.reflect.KProperty<R>` is a subtype of `kotlin.reflect.KCallable<R>`. Other members or base types for this class may be provided by platform and/or implementation.

3.16.4 `kotlin.reflect.KFunction`

`kotlin.reflect.KFunction<out R>` is the class used to represent runtime information for functions. It is the base type of function references. `kotlin.reflect.KFunction<R>` is a subtype of `kotlin.reflect.KCallable<R>` and `kotlin.Function<R>`. Other members or base types for this class may be provided by platform and/or implementation.