Kotlin

Kotlin doesn't have static methods, so they should be place inside a [companion object](https://kotlinlang.org/docs/reference/object-declarations.html#companion-objects). If you still want to use them from Java code, annotate them with [@JvmStatic](https://kotlinlang.org/docs/reference/java-to-kotlin-interop.html#static-methods).

**NOTE**: object declarations can't be local (i.e. be nested directly inside a function), but they can be nested into other object declarations or non-inner classes.

Companion Objects

An object declaration inside a class can be marked with the companion keyword:

**class** MyClass {

**companion** **object** Factory {

**fun** create(): MyClass = MyClass()

}

}

Members of the companion object can be called by using simply the class name as the qualifier:

**val** instance = MyClass.create()

The name of the companion object can be omitted, in which case the name Companion will be used:

**class** MyClass {

**companion** **object** {

}

}

**val** x = MyClass.Companion

Note that, even though the members of companion objects look like static members in other languages, at runtime those are still instance members of real objects, and can, for example, implement interfaces:

**interface** Factory<T> {

**fun** create(): T

}

**class** MyClass {

**companion** **object** : Factory<MyClass> {

**override** **fun** create(): MyClass = MyClass()

}

}

However, on the JVM you can have members of companion objects generated as real static methods and fields, if you use the @JvmStatic annotation. See the [Java interoperability](https://kotlinlang.org/docs/reference/java-to-kotlin-interop.html#static-fields) section for more details.

Semantic difference between object expressions and declarations

There is one important semantic difference between object expressions and object declarations:

* object expressions are executed (and initialized) **immediately**, where they are used;
* object declarations are initialized **lazily**, when accessed for the first time;
* a companion object is initialized when the corresponding class is loaded (resolved), matching the semantics of a Java static initializer.