# **Interfaces**

Interfaces in Kotlin are very similar to Java 8. They can contain declarations of abstract methods, as well as method implementations. What makes them different from abstract classes is that interfaces cannot store state. They can have properties but these need to be abstract or to provide accessor implementations.

An interface is defined using the keyword interface

**interface** MyInterface {

**fun** bar()

**fun** foo() {

// optional body

}

}

## Implementing Interfaces

A class or object can implement one or more interfaces

**class** Child : MyInterface {

**override** **fun** bar() {

// body

}

}

## Properties in Interfaces

You can declare properties in interfaces. A property declared in an interface can either be abstract, or it can provide implementations for accessors. Properties declared in interfaces can't have backing fields, and therefore accessors declared in interfaces can't reference them.

**interface** MyInterface {

**val** prop: Int // abstract

**val** propertyWithImplementation: String

**get**() = "foo"

**fun** foo() {

print(prop)

}

}

**class** Child : MyInterface {

**override** **val** prop: Int = 29

}

## Resolving overriding conflicts

When we declare many types in our supertype list, it may appear that we inherit more than one implementation of the same method. For example

**interface** A {

**fun** foo() { print("A") }

**fun** bar()

}

**interface** B {

**fun** foo() { print("B") }

**fun** bar() { print("bar") }

}

**class** C : A {

**override** **fun** bar() { print("bar") }

}

**class** D : A, B {

**override** **fun** foo() {

**super**<A>.foo()

**super**<B>.foo()

}

**override** **fun** bar() {

**super**<B>.bar()

}

}

Interfaces A and B both declare functions foo() and bar(). Both of them implement foo(), but only Bimplements bar() (bar() is not marked abstract in A, because this is the default for interfaces, if the function has no body). Now, if we derive a concrete class C from A, we, obviously, have to override bar() and provide an implementation.

However, if we derive D from A and B, we need to implement all the methods which we have inherited from multiple interfaces, and to specify how exactly D should implement them. This rule applies both to methods for which we've inherited a single implementation (bar()) and multiple implementations (foo()).