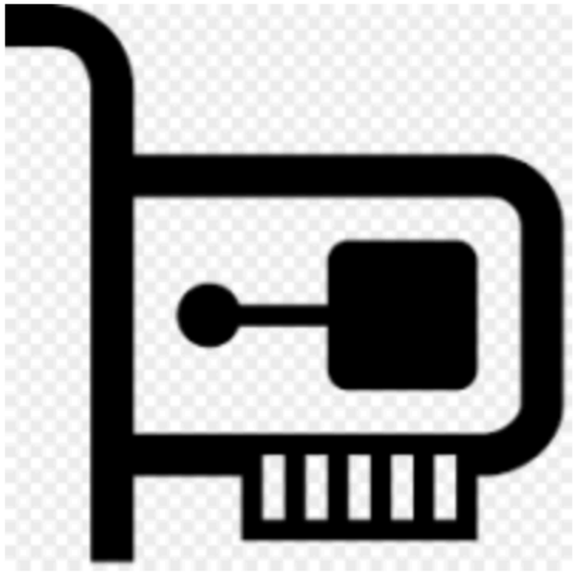
# Interfaces

## Interfaces



Kotlin interfaces largely follow Java 8 conventions

# 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
}
```

# Interfaces Inheritance

An interface can derive from other interfaces and thus both provide implementations for their members and declare new functions and properties. Quite naturally, classes implementing such an interface are only required to define the missing implementations:

```
interface Named {
    val name: String
}

interface Person : Named {
    val firstName: String
    val lastName: String

    override val name: String get() = "$firstName $lastName"
}

data class Employee(
    // implementing 'name' is not required
    override val firstName: String,
    override val lastName: String,
    val position: Position
) : Person
```