Conventions

Comparisons

```
"abc" < "def"
```

```
a >= b
```

→

```
a.compareTo(b) >= 0
```

Comparisons

symbol	translated to
`a > b`	`a.compareTo(b) > 0`
`a < b`	`a.compareTo(b) < 0`
`a >= b`	`a.compareTo(b) >= 0`
`a <= b`	`a.compareTo(b) <= 0`

Equality check

```
s1 == s2
```

Calls `equals` under the hood:

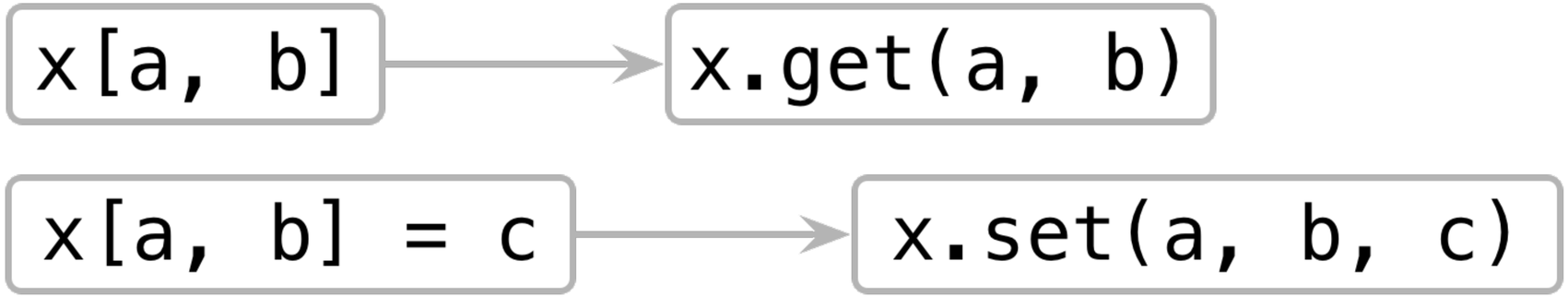
```
s1.equals(s2)
```

Correctly handles nullable values:

```
s1 == s2          // s1.equals(s2)
null == "abc"     // false
null == null      // true
```

Accessing elements by index: `[]`

```
map[key]
mutableMap[key] = newValue
```



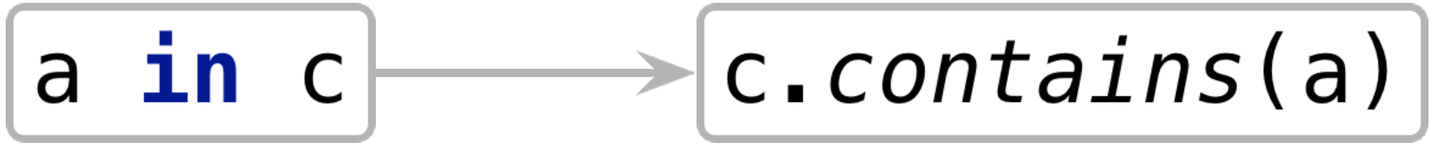
Accessing elements by index: []

```
interface Map<K, V> {
    operator fun get(key: K): V?
}

operator fun <K, V> MutableMap<K, V>.set(key: K, value: V) {
    put(key, value)
}
```

The `in` convention

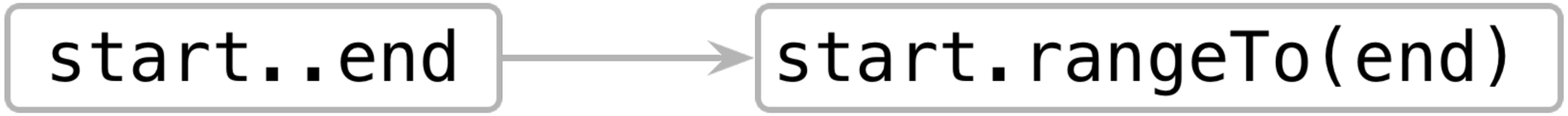
```
if (key in map) { }
if (element in list) { }
```



The `rangeTo` convention

```
if (s in "abc".."def") { }
for (i in 1..2) { }

val oneTo100: IntRange = 1..100
for (i in oneTo100) { }
```



The `iterator` convention

```
operator fun CharSequence.iterator(): CharIterator

for (c in "abc") { }
```

Destructuring declarations

```
val (first, second) = pair
for ((key, value) in map) { }
map.forEach { (key, value) -> }
```

```
val (a, b) = p
```

→

```
val a = p.component1()
val b = p.component2()
```

Destructuring declarations & data classes

```
data class Contact(
        val name: String,
        val email: String,
        val phoneNumber: String
)
```

```
val (name, _, phoneNumber) = contact
```

Destructuring declarations & data classes

```
data class Contact(
        val name: String,
        val email: String,
        val phoneNumber: String
) {
```

generated methods

```
    fun component1() = name
    fun component2() = email
    fun component3() = phoneNumber
}
```

```
val (name, _, phoneNumber) = contact
```

Which elements can be compared using comparison operations?

```
x < y
x >= y
```

1. all primitives like `Int`, `Double` and `Boolean`
2. `String`s
3. elements implementing `Comparable` interface
4. elements that define member or extension `operator` function `compareTo` (with the right signature)

?

Which elements can be compared using comparison operations?

```
x < y
x >= y
```

1. all primitives like `Int`, `Double` and ~~`Boolean`~~
2. `Strings`
3. elements implementing `Comparable` interface
4. elements that define member or extension `operator` function `compareTo` (with the right signature)

`compareTo` convention

```
operator fun Point.compareTo(other: Point): Int {
    ...
}
```

Conventions & Java

```
package java.lang;

public interface Comparable<T> {
    public int compareTo(T o);
}
```

Operator syntax works for all Java methods with the right signature

Extensions might be added

```
import java.math.BigInteger

operator fun BigInteger.plus(other: BigInteger) =
    this.add(other)


BigInteger.TEN + BigInteger.ONE
```