Nullable types under the hood

Nullable Types Under the Hood

`@Nullable`, `@NotNull` annotations

No performance overhead

Nullable types ≠ Optional

```
class Optional<T>(val value: T) {

    fun isPresent() = value != null

    fun get() = value ?:
            throw NoSuchElementException("No value present")
}
```

How many objects are created to store a value of a nullable String?

```
val s: String?
```

1. Two: one object to store a `String` value, another (a wrapper) to store a nullable String
2. Only one object to store a `String` value

?

How many objects are created to store a value of a nullable String?

```
val s: String?
```

1. Two: one object to store a `String` value, another (a wrapper) to store a nullable String
2. Only one object to store a `String` value

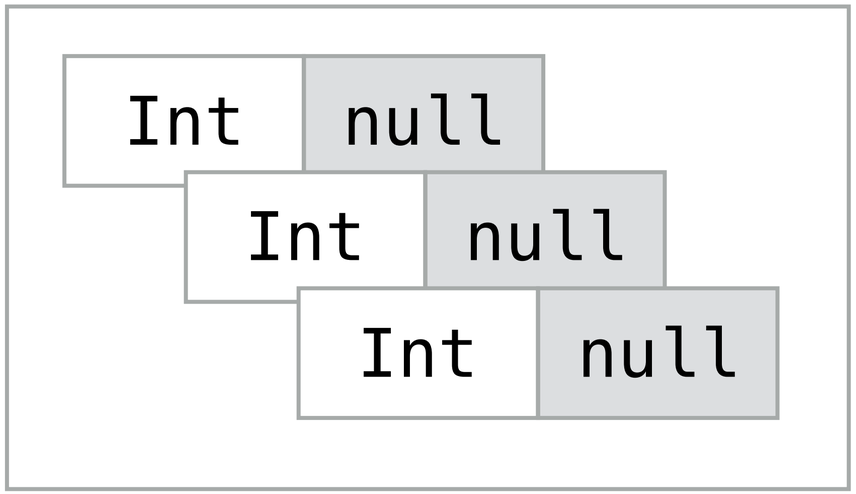
Under the hood

```kotlin
fun foo(): String = "foo"
fun bar(): String? = "bar"
```

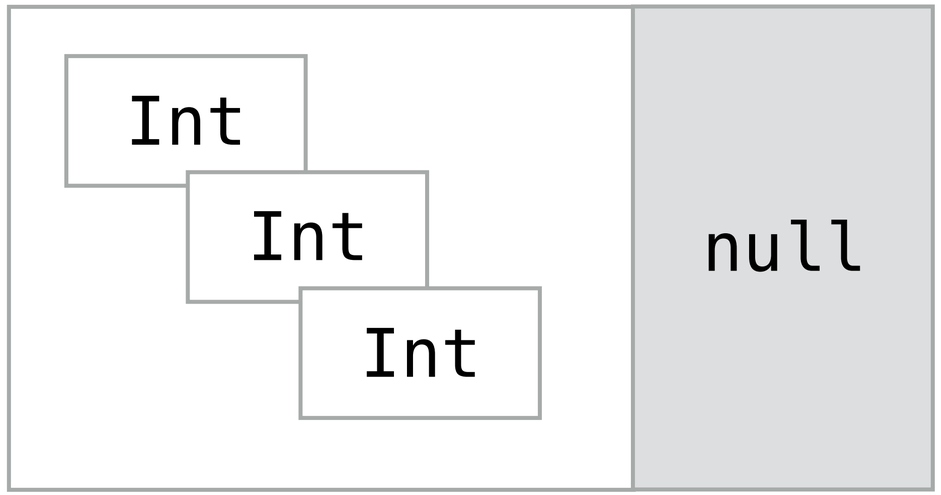
```java
@NotNull
public static final String foo() {
    return "foo";
}

@Nullable
public static final String bar() {
    return "bar";
}
```

List of nullable elements vs nullable list



List<Int?>



List<Int>?

Mark the lines which require a question mark to make the code compile

```
    fun foo(list1: List<Int?>, list2: List<Int>?) {
#1      list1.size
#2      list2.size

#3      val i: Int =
#4              list1.get(0)

#5      val j: Int =
#6              list2.get(0)
    }
```

?

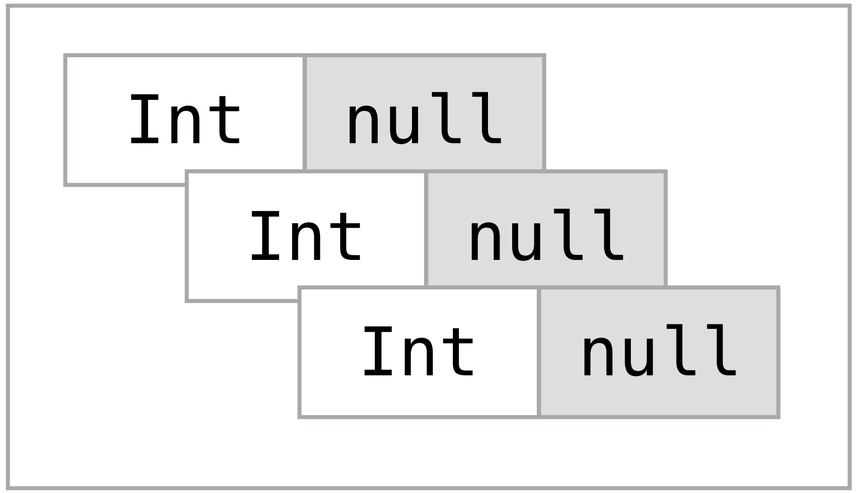
Mark the lines which require a question mark to make the code compile

```
    fun foo(list1: List<Int?>) {

#1      list1.size

#3      val i: Int =
#4              list1.get(0)
    }
```

Compiler error: Type mismatch:
inferred type is Int? but Int was expected

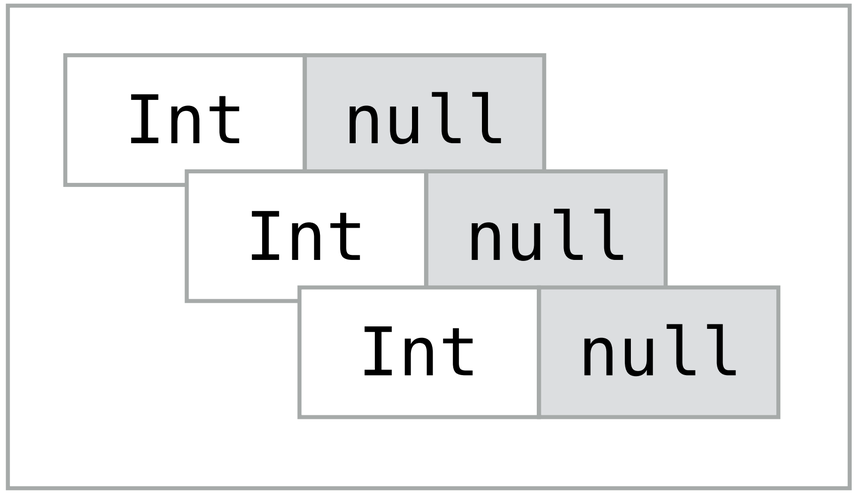


Mark the lines which require a question mark to make the code compile

```
    fun foo(list1: List<Int?>) {

#1      list1.size

#3      val i: Int? =
#4                 list1.get(0)
    }
```

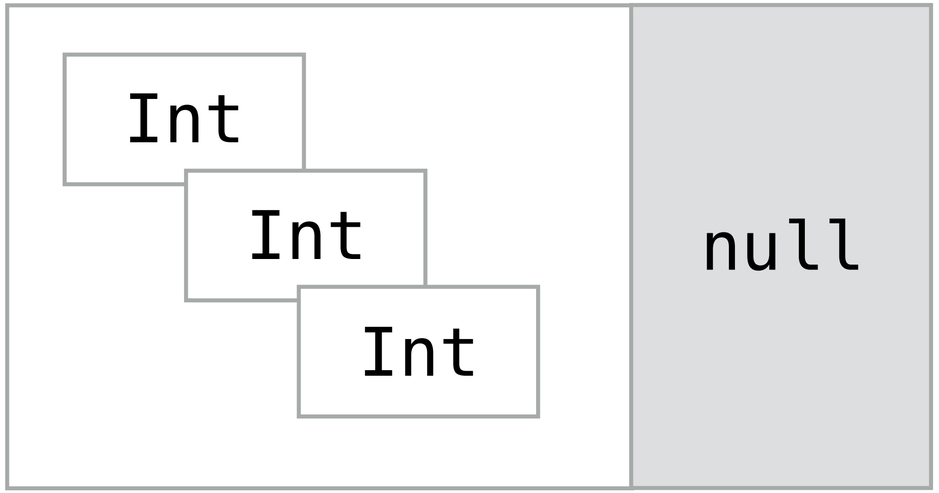


Mark the lines which require a question mark to make the code compile

```
    fun foo(list2: List<Int>?) {

#2      list2.size

#5      val i: Int =
#6              list2.get(0)
    }
```



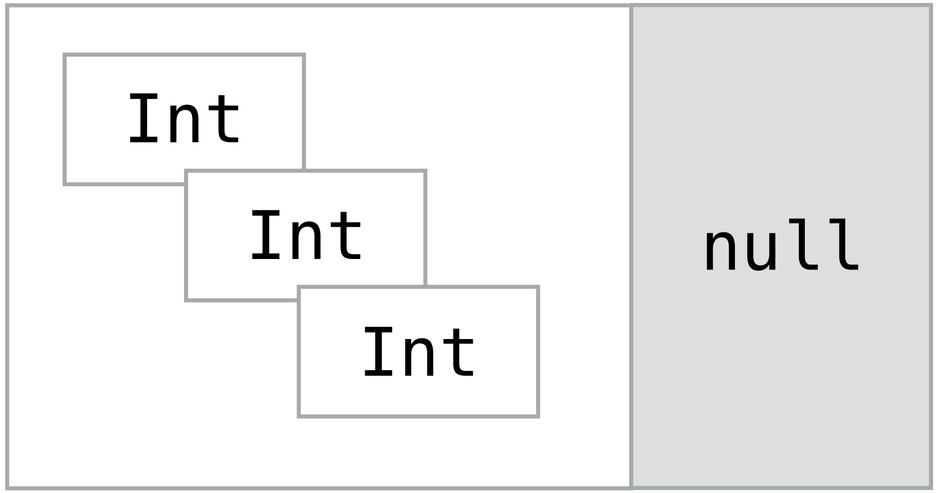
Compiler errors: Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type List<Int>?

Mark the lines which require a question mark to make the code compile

```
    fun foo(list2: List<Int>?) {

#2      list2?.size

#5      val i: Int =
#6              list2?.get(0)
    }
```



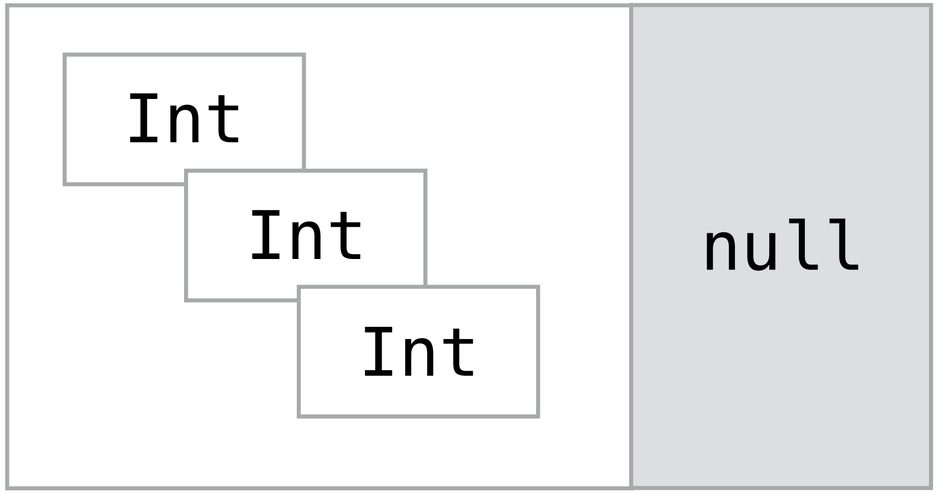
Mark the lines which require a question mark to make the code compile

```
    fun foo(list2: List<Int>?) {

#2      list2?.size

#5      val i: Int =
#6              list2?.get(0)
    }
```

Compiler error: Type mismatch:
inferred type is Int? but Int was expected

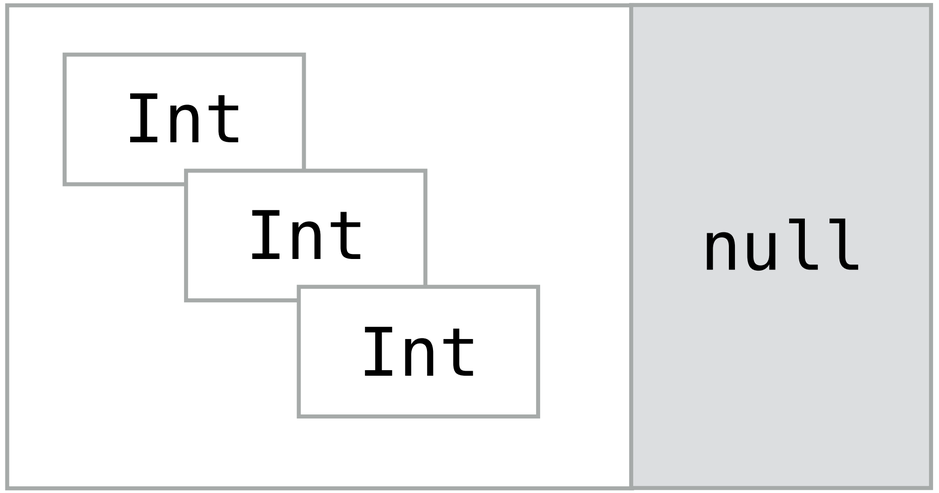


Mark the lines which require a question mark to make the code compile

```
    fun foo(list2: List<Int>?) {

#2      list2?.size

#5      val i: Int? =
#6                 list2?.get(0)
    }
```



Mark the lines which require a question mark to make the code compile

```
    fun foo(list1: List<Int?>, list2: List<Int>?) {
#1      list1.size
#2      list2?.size

#3      val i: Int? =                          #2, #3, #5, #6
#4              list1.get(0)

#5      val j: Int? =
#6              list2?.get(0)
    }
```