Kotlin & C#

A Comparison of Two Modern Languages

Kirill Rakhman

Syntax

Properties

Kotlin

```
val immutable: String = "Hello"

var mutable: String = "World"

val computed: String get() = "!"
```

C#

```
public string Immutable { get; } = "Hello";

public string Mutable { get; set; } = "World";

public string Computed => "!";
```

Classes & Constructors

Kotlin

```
class Foo(
    val bar: String,
    val baz: Int
)
```

C#

```
public class Foo
{
   public string Bar { get; }

   public int Baz { get; }

   public Foo(string bar, int baz)
   {
       Bar = bar;
       Baz = baz;
   }
}
```

Class Instantiation

Kotlin

```
class Foo(
    val bar: String,
    val baz: Int
)

val foo = Foo("A String", 42)
```

C#

```
public class Foo
{
    ...
}

var foo = new Foo("A String", 42);
```

Class Initialization Syntax

Kotlin

```kotlin
class Foo(
    val bar: String,
    val baz: Int
)


val foo = Foo(
   bar = "A String",
   baz = 42
)
```

C#

```csharp
public class Foo
{
   public string Bar { get; set; }
   public int Baz { get; set; }
}

Foo foo = new Foo
{
   Bar = "A String",
   Baz = 42
};
```

Primary Constructors

Kotlin

```
class Foo(
    val bar: String,
    val baz: Int
)
```

C#

```
public class Foo(string bar, int baz)
{
   public string Bar { get; } = bar;

   public int Baz { get; } = baz;
}
```

Planned for C# 6. Removed

Data Classes / Records

Kotlin

```
data class Foo(
   val bar: String, val baz: Int)

val foo = Foo("A String", 42)

foo.copy(bar = "Another String")
```

C#

```
public class Foo(string Bar, int Baz);


var foo = new Foo("A String", 42);

foo.With(Bar: "Another String")
```

Planned for C# 7. Postponed

Weird Tuple Hack

C#

```
public class Person
{
   public string Name { get; }
   public int Age { get; }

   public Person(string name, int age) => (Name, Age) = (name, age);
}
```

Scoping and Pattern Matching

Let / Out Variables

Kotlin

```
val map = mapOf<String, String>()

map["key"]?.let { value ->


   println(value)
}
```

C#

```
Dictionary<string, string> dictionary = ...

if (dictionary
        .TryGetValue("key", out string value))
{
    Console.WriteLine(value);
}
```

Let / Var Pattern

Kotlin

```
fun getValue(): String? { … }

getValue()?.let { value ->

    println(value)
}
```

C#

```
public string? GetValue() { … }

if (GetValue() is string value)
{
    Console.WriteLine(value);
}
```

Out Variables & Pattern Matching

Kotlin

```
val map = mapOf<String, Any>()

(map["key"] as? String)?.let { value ->



    println(value)
}
```

C#

```
Dictionary<string, object> dictionary = ...

if (dictionary
        .TryGetValue("key", out object value)
&& value is string s)
{
    Console.WriteLine(s);
}
```

When / Switch

Kotlin

```kotlin
fun format(foo: Any): String {

    return when(foo) {

        "0" -> "Zero"
        is String -> foo.toUpperCase()
        is Pair<*,*> ->
             "(${foo.first}, ${foo.second})"
        else -> foo.toString()
   }
}
```

C#

```csharp
public string Format(object foo)
{
   return foo switch
   {
       "0" => "Zero",
       string s => s.ToUpper(),
       (string a, string b) =>
                       $"({a}, {b})",
       _ => foo.ToString()
   };
}
```

Advanced Pattern Matching

C#

```
static string Display(object o) => o switch
{
   Point { X: 0, Y: 0 }           p => "origin",
   Point { X: var x, Y: var y } p => $"({x}, {y})",
   _                                => "unknown"
};
```

Type System

Nullable Reference Types

Kotlin

```
class Foo {

   var bar: String? = null

   fun doThings(thing: Thing?) {

      thing?.call()
      if (thing != null) thing.call()

      if (bar != null) bar.split("")
   }
}
```

C#

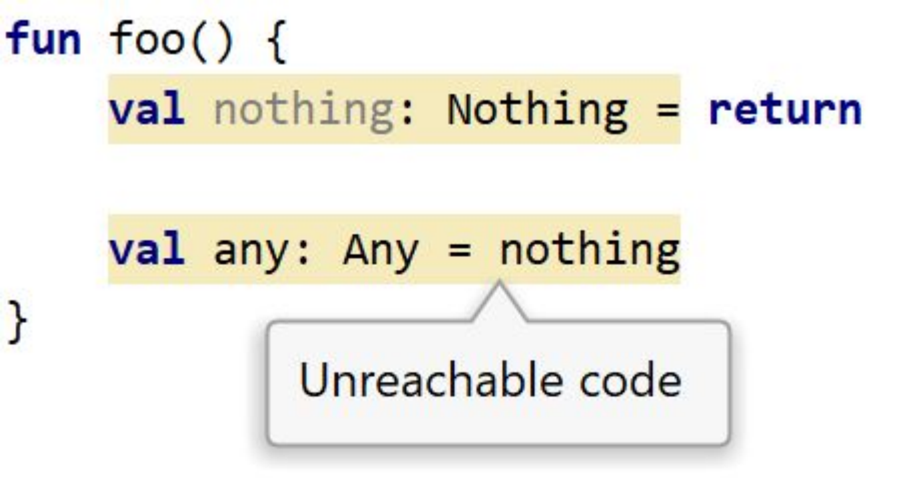
```
class Foo
{
   string? bar;

   void DoThings(Thing? thing)
   {
      thing?.Call();
      if (thing != null) thing.Call();

      if (bar != null) bar.Split("");
   }
}
```

Nullable Structs

C#

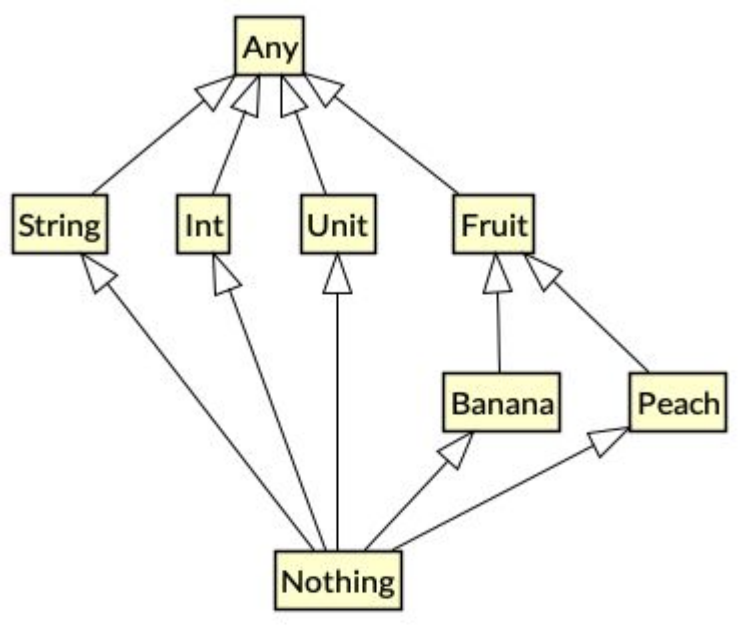
```
public struct Nullable<T> where T : struct
{
    private T value;
    public bool HasValue { get; }
}

Nullable<int> foo = new Nullable<int>();
if (foo.HasValue) { ... }

int? bar = null;
if (bar != null) { … }
```

Refresher: Nothing Type



```
fun foo() {
    val nothing: Nothing = return

    val any: Any = nothing
}
```

Unreachable code



http://www.natpryce.com/articles/000818.html

Nothing Typed Operators

Kotlin

```kotlin
val nullableVariable: String? = null;

val value: String = nullableVariable
    ?: throw Exception()

val value: String = nullableVariable ?: return
val value: String = nullableVariable ?: break
val value: String = nullableVariable ?: continue

val value: String = nullableVariable
    ?: exitProcess(0)
```

C#

```csharp
public Foo(string? bar)
{
  this.bar = bar
        ?? throw new ArgumentNullException();
}
```

Function Types, Lambdas, Method References

Kotlin

```kotlin
fun foo(f: (Int) -> String) { … }

fun intToString(x: Int)
    = "Number: $x"

foo(::intToString)
foo { it.toString() }

foo(Int::toString)
```

C#

```csharp
delegate string Format(int message);

static void Foo(Format f) { … }

static string IntToString(int x)
    => $"Number: {x}";

Foo(IntToString);
Foo(x => x.ToString());
```

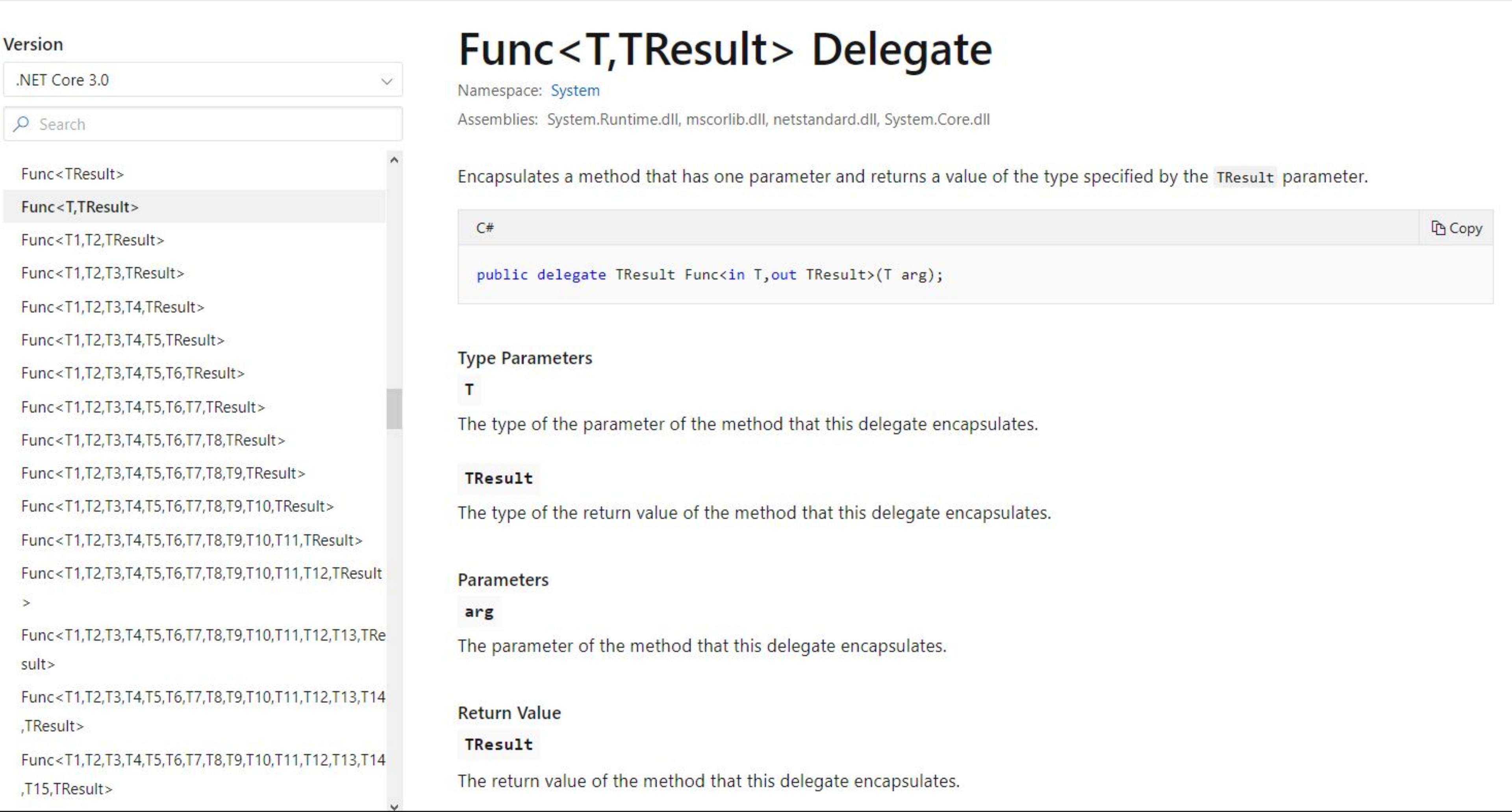
Function Types, Lambdas, Method References

Kotlin

```kotlin
fun foo(f: (Int) -> String) { … }

fun intToString(x: Int)
    = "Number: $x"

foo(::intToString)
foo { it.toString() }

foo(Int::toString)
```

C#

```csharp
static void Foo(Func<int, string> f) {}

static string IntToString(int x)
    => $"Number: {x}";

Foo(IntToString);
Foo(x => x.ToString());
```


Version
.NET Core 3.0
Search
Func<TResult>
Func<T,TResult>
Func<T1,T2,TResult>
Func<T1,T2,T3,TResult>
Func<T1,T2,T3,T4,TResult>
Func<T1,T2,T3,T4,T5,TResult>
Func<T1,T2,T3,T4,T5,T6,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,TResult>
Func<T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,TResult>
Copy

Func<T,TResult> Delegate

Namespace: System

Assemblies: System.Runtime.dll, mscorlib.dll, netstandard.dll, System.Core.dll

Encapsulates a method that has one parameter and returns a value of the type specified by the `TResult` parameter.

C#

```
public delegate TResult Func<in T,out TResult>(T arg);
```

Type Parameters

`T`

The type of the parameter of the method that this delegate encapsulates.

`TResult`

The type of the return value of the method that this delegate encapsulates.

Parameters

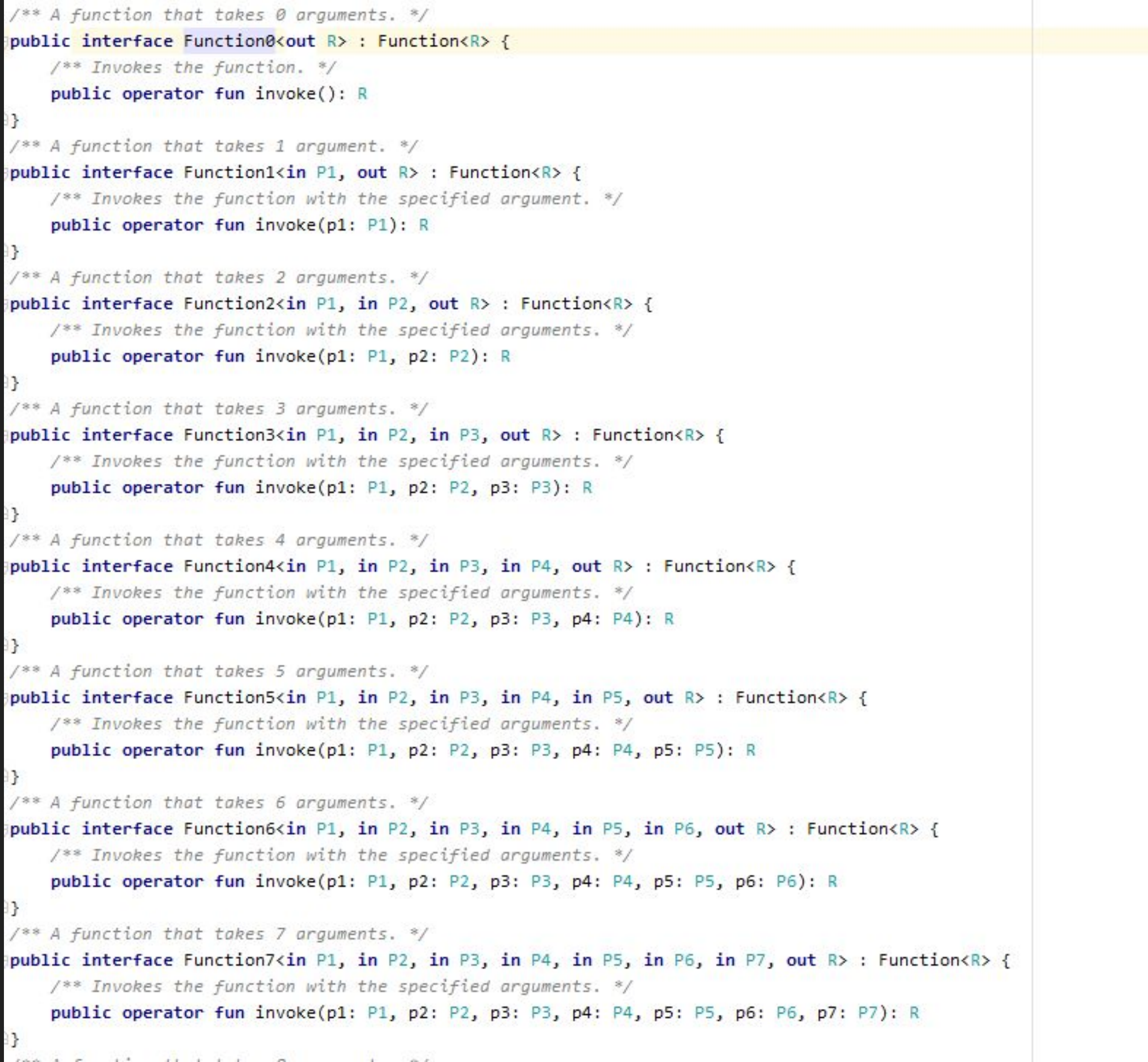
`arg`

The parameter of the method that this delegate encapsulates.

Return Value

`TResult`

The return value of the method that this delegate encapsulates.



```kotlin
/** A function that takes 0 arguments. */
public interface Function0<out R> : Function<R> {
    /** Invokes the function. */
    public operator fun invoke(): R
}
/** A function that takes 1 argument. */
public interface Function1<in P1, out R> : Function<R> {
    /** Invokes the function with the specified argument. */
    public operator fun invoke(p1: P1): R
}
/** A function that takes 2 arguments. */
public interface Function2<in P1, in P2, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2): R
}
/** A function that takes 3 arguments. */
public interface Function3<in P1, in P2, in P3, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2, p3: P3): R
}
/** A function that takes 4 arguments. */
public interface Function4<in P1, in P2, in P3, in P4, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2, p3: P3, p4: P4): R
}
/** A function that takes 5 arguments. */
public interface Function5<in P1, in P2, in P3, in P4, in P5, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2, p3: P3, p4: P4, p5: P5): R
}
/** A function that takes 6 arguments. */
public interface Function6<in P1, in P2, in P3, in P4, in P5, in P6, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2, p3: P3, p4: P4, p5: P5, p6: P6): R
}
/** A function that takes 7 arguments. */
public interface Function7<in P1, in P2, in P3, in P4, in P5, in P6, in P7, out R> : Function<R> {
    /** Invokes the function with the specified arguments. */
    public operator fun invoke(p1: P1, p2: P2, p3: P3, p4: P4, p5: P5, p6: P6, p7: P7): R
}
```

Events

C#

```
public delegate void EventHandler(object sender, EventArgs e);

public event EventHandler ThresholdReached; // no initializer

ThresholdReached += (sender, e) => { … }


void OnThresholdReached(EventArgs e)
{
   // Watch out for race conditions
   EventHandler handler = ThresholdReached;
   if (handler != null)
   {
      handler.Invoke(this, e);
   }
}
```

Events

C#

```
public delegate void EventHandler(object sender, EventArgs e);

public event EventHandler ThresholdReached;

ThresholdReached += (sender, e) => { … }


void OnThresholdReached(EventArgs e)
{
   ThresholdReached?.Invoke(this, e);
}
```

Asynchronicity

Couroutines / Async Await

Kotlin

```
suspend fun getFoo(): String {

   val s = bar()
   return s.toUpperCase()
}

suspend fun bar()
    = "Hello"
```

C#

```
async Task<string> GetFooAsync()
{
   var s = await BarAsync();
   return s.ToUpper();
}

Task<string> BarAsync() =>
    Task.FromResult("Hello");
```

Asynchronous Branching

Kotlin

```
val deferred = async {
    getFoo()
}

// do things

val foo = deferred.await()
```

C#

```
Task task = GetFooAsync();



// do things

var foo = await task;
```

Async Parallelism

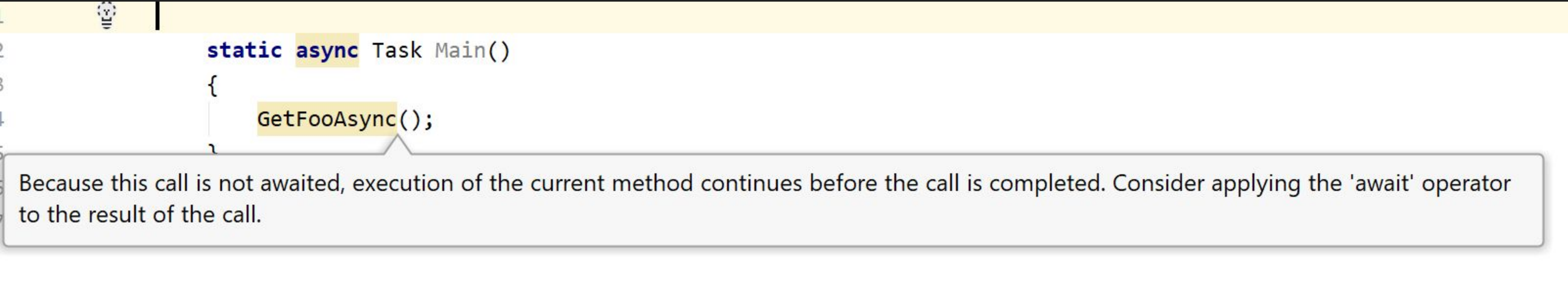
Kotlin

```
val results = awaitAll(
    async { getFoo() },
    async { getBar() })
```

C#

```
var results = await Task.WhenAll(
    GetFooAsync(),
    GetBarAsync());
```

Forgetting to call await



```
static async Task Main()
{
    GetFooAsync();
}
```

Because this call is not awaited, execution of the current method continues before the call is completed. Consider applying the 'await' operator to the result of the call.

Cancellation

Kotlin

```
val job = launch { doThings() }

job.cancel()


suspend fun doThings() {
   delay(100)
   coroutineScope { launch {} }
   yield()
   // or
   if (!isActive) return
}
```

C#

```
using var tokenSource =
    new CancellationTokenSource();

await DoThingsAsync(tokenSource.Token);

tokenSource.Cancel();


async Task DoThingsAsync(
    CancellationToken token)
{
   token.ThrowIfCancellationRequested();
   // or
   if (token.IsCancellationRequested) return
}
```

Thx for Listening

@Cypressious

rakhman.info