

15. What is the difference between typeof and GetType?

Brief summary: Both **typeof** keyword and the **GetType** method are used to get the information about some type. The differences between them are:

- **typeof** takes the name of the type we want to inspect, so we must know the type before. **typeof** is resolved at compile time.
- **GetType** is a method that must be executed on an object. Because of that, it is resolved at runtime. This method comes from the System.Object base class, so it is available in any object in C#

Both **typeof** keyword and the **GetType** method are used to get the information about some type. The differences between them are:

- **typeof** takes the name of the type we want to inspect, so we must know the type before. **typeof** is resolved at compile time.
- **GetType** is a method that must be executed on an object. Because of that, it is resolved at runtime. This method comes from the System.Object base class, so it is available in any object in C#

Let's see this in practice. If I know the type already, and I only want to get the Type object for some reason, **typeof** and the **GetType** method will give me the same result:

```
var type1 = typeof(Base);
Console.WriteLine(type1.FullName);

var baseObj = new Base();
var type2 = baseObj.GetType();
Console.WriteLine(type2.FullName);
```

In the first case, I use the **typeof** with the Base type. In the second, I execute the **GetType** method on an object of this type. In both cases, the result is the Type object containing full information about the Base type.

```
type1: Base
type2: Base
```

But I don't always know the type at compile time. Let's consider this code:

```
void PrintTypeName(object obj)
{
    var type = obj.GetType();
    Console.WriteLine("type name is: " + type.FullName);
}
```

The obj may be anything, and it will only be known at runtime what it is exactly. That's why we can't use **typeof** here. We should rather use **GetType**. Let's see this method in action:

```
Derived derived = new Derived();
PrintTypeName(derived);

string text = "abc";
PrintTypeName(text);
```

```
type name is: Derived
type name is: System.String
```

The important thing to understand is that `GetType` always returns the actual type of an object. Let's consider this code:

```
Base derivedAsBase = new Derived();
var type4 = derivedAsBase.GetType();
Console.WriteLine("type4: " + type4.FullName);
```

Even if the variable **derivedAsBase** is of type `Base`, we assign an object of type `Derived` to it. It is possible because `Derived` inherits from `Base`. The `GetType` method will print the actual type:

```
type4: Derived
```

We actually have seen this before, in the `PrintTypeName` method. Even if it took a parameter of type `System.Object`, it printed the actual type of the given object. Remember that the `GetType` method belongs to `System.Object` type, so it can be called for any object in C#.

Let's summarize. Both **`typeof`** and the **`GetType`** method return a `Type` object, which holds the information about a type. **`typeof`** takes the name of the type, and it gets resolved at compile time. The `GetType` method is called upon an object, so it is resolved at runtime.

Both **`typeof`** and the **`GetType`** method are parts of the reflection mechanism, which we will learn about in the next lecture.

Bonus questions:

- **"What is the purpose of the `GetType` method?"**
This method returns the `Type` object which holds all information about the type of the object it was called on. For example, it contains the type name, list of the constructors, attributes, the base type, etc.
- **"Where is the `GetType` method defined?"**
It is defined in the `System.Object` type, which is a base type for all types in C#. This is why we can call the `GetType` method on objects of any type.