Reflection

.NET

Reflection assists languages to operate in networks by enabling libraries for serialization, bundling, and varying data formats.
Reflection a language more suited to network-oriented code.

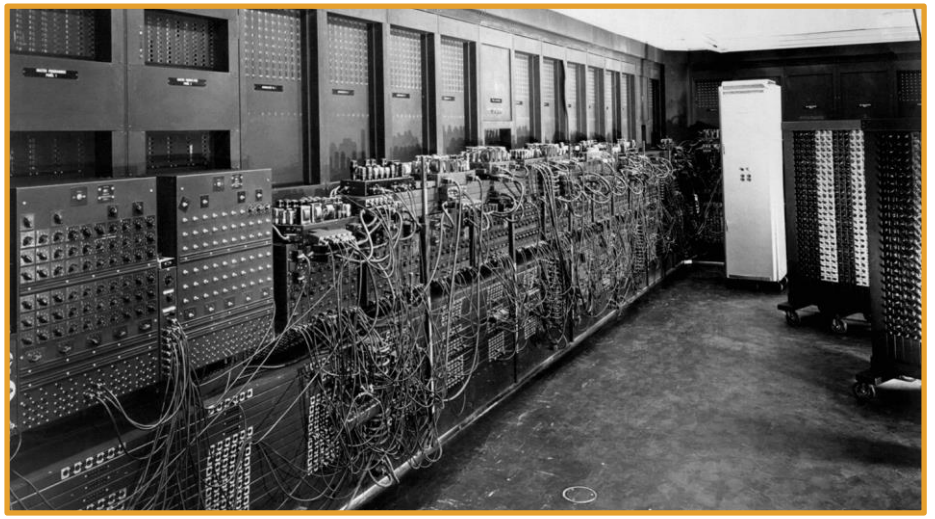
https://en.wikipedia.org/wiki/Reflective_programming

A History of Reflection.

The earliest computers were inherently reflective. Their architectures were programmed by defining instructions as data and using self-modifying code.

As programming moved to higher-level, compiled languages (C,C++) this reflective ability largely disappeared until new programming languages with built-in reflection appeared.

In 1982, Brian Smith proposed computational reflection in procedural programming languages and the notion of the meta-circular interpreter as a component of 3-Lisp.



What is Reflection?

https://en.wikipedia.org/wiki/Reflective_programming

Reflection can be used for observing and modifying program execution during runtime based on the type and status of different program elements. A reflection-oriented program component can monitor the execution of code and then modify itself according to predefined goals.

In OOP languages, reflection:

- allows inspection of classes, interfaces, fields and methods at runtime without knowing the names of the interfaces, fields, methods at compile time.
- allows instantiation of new objects and invocation of methods.
- is used as part of testing for mocking objects and bringing otherwise unreachable, and thereby untestable, code into scope.

In some languages, reflection can be used to bypass member accessibility rules. For C# properties, this can be achieved by writing directly onto the (usually invisible) backing field of a non-public property. It is also possible to find non-public class methods and manually invoke them.

Reflection in .NET

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection

Reflection provides type Type objects that describe assemblies, modules, and types.

You can use reflection to:

- dynamically create an instance of a type,
- bind the type to an existing object, or
- get the type from an existing object and invoke its methods or access its fields and properties.

```
// Using GetType() to obtain
type information:
int myInt = 42;
Type type = myInt.GetType();
Console.WriteLine(type);

/* prints
System.Int32
 */
```

When to use Reflection

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection#reflection-overview
https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection

---

Reflection is useful in the following situations:

- When you need to access attributes in your program's metadata.
  - Retrieving Information Stored in Attributes.
- For examining and instantiating Types in an assembly.
- Use System.Reflection.Emit for building new types at run time.
- For performing late binding, accessing methods on types created at run time.
  - See Dynamically Loading and Using Types.

```
// Without reflection
Foo foo = new Foo();
foo.PrintHello();

// With reflection
Object foo =
Activator.CreateInstance("complete.classpath.and.Foo");
MethodInfo method = foo.GetType().GetMethod("PrintHello");
method.Invoke(foo, null);
```

Type Class, Reflections datatype

https://docs.microsoft.com/en-us/dotnet/api/system.type?view=net-6.0