Reflection

Looking inside yourself

Contents

- What is reflection?
- Diagrams
- Equations
- Code Samples
- Summary
- References

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

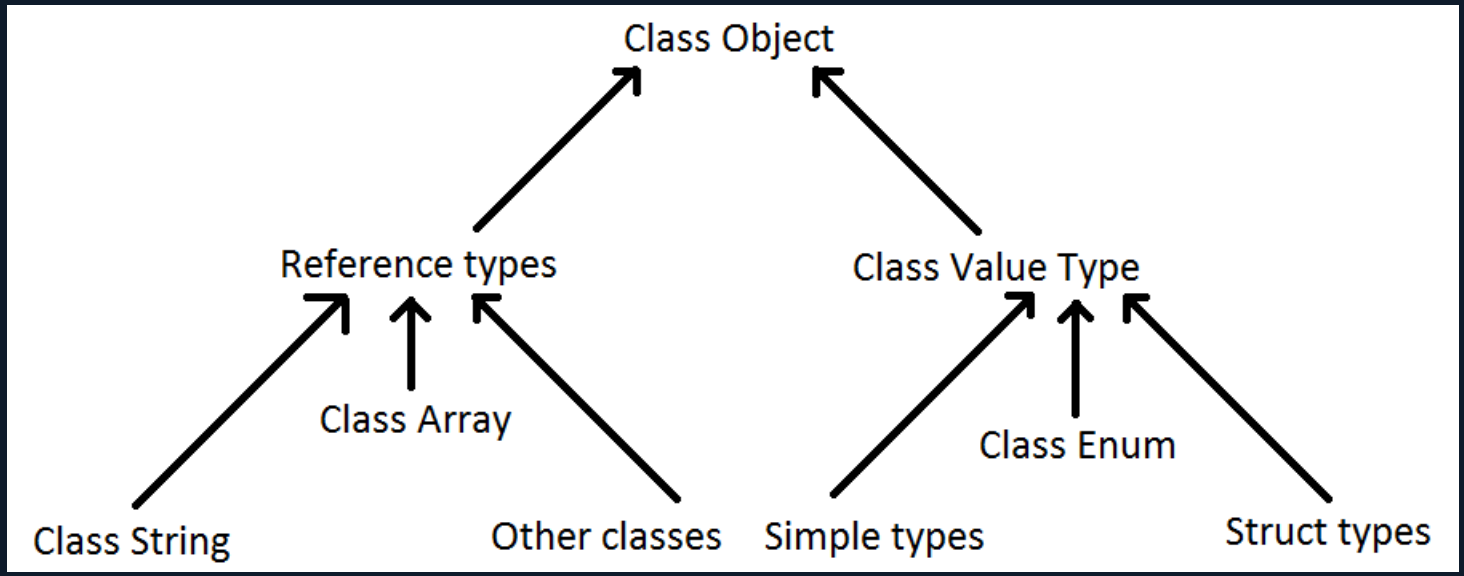
What is Reflection?

- Reflection is the ability of a language to read its own metadata
- You can fetch type information at runtime programmatically
- Allows dynamic loading of assemblies compiled for other platforms

Objects and Types

- Everything in C# is an Object (Class), both predefined and user defined
- All data types directly or indirectly inherit from the Object class
- We will be using Object types later
- The type class represents all type declarations
- Allows access to metadata for Reflection

Overall class hierarchy from C#



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

The Type Class

- Primary way to get access to metadata of a type
- Includes classes, structures, enumerations, interfaces, delegates
- Describes how types are declared, used and managed in the common language runtime

```
Car c = new Car();
Type t = c.GetType();
```

- Above, t contains information about the Car class, all the members and members inside the class.

Getting Type Information

- There are three main ways:
  - System.Object.GetType()
  - System.Type.GetType()
  - C# Operator typeof

C# Operator Typeof

- The simplest way to get Type information

- Similar to C++ typeof

```
Type t = typeof(Car);
```

- The variable t now contains information about the Type that Car is

System.Object.GetType()

- This method gets an instance of an object to report what Type it is

```
Car c = new Car();
Type t = c.GetType();
```

- Need to have knowledge of the type beforehand

System.Type.GetType()

- Has a few overloads, check MSDN for further information about what each does

```
Type t = Type.GetType("Reflection.Car", false, true);
```

- One specific overload takes in a string of the class name, a Boolean to specify case sensitivity and a Boolean to specify throwing exceptions for unknown types

What's in the type class?

- Broken down into Properties and Methods

- Properties: A mechanism for reading and writing values, similar to getters and setters in C++, provides a simple way to get both get and set functionality

Properties of the Type class

- Falls into three categories
  - Strings containing various names associated with the class e.g. Name, FullName
  - Objects that the class is related to, e.g. BaseType, UnderlyingSystemType
  - Boolean properties, e.g. is the type abstract, a class, an array, a pointer, an enum, etc.

Methods of the Type class

- Otherwise known as functions, these are used to obtain details of the members of the class
- Each method returns a different data type, but method names are good enough to give information on what they do
- E.g. GetConstructor / GetConstructors return ConstructorInfo

Using methods of the Type class

- Can be used to get specific information about a type
- GetMethod() returns a reference to a MethodInfo object containing information about a specific method
- GetMethods() returns an array of MethodInfo objects containing information about all the methods within that type

Using methods of the Type class

- Both methods have overloads to take in an extra parameter which specifies which members should be returned based on certain flags

- For example:
  - Only return public methods that are static
  - Only return private methods that are not static

Example

```
Car c = new Car();
Type t = c.GetType();
MethodInfo carMethodInfo = t.GetMethod("IsMoving");
Console.WriteLine(carMethodInfo.Name);

MethodInfo[] carMethods = t.GetMethods();
foreach (MethodInfo mi in carMethods)
{
    Console.WriteLine(mi.Name);
}
```

Fields and member variables

- Interacting with constructors, events, fields, properties and members of a class are done in a similar way

- Fields and member variables can be changed and manipulated at run time without ever knowing about the class type beforehand

Activator Class

- Need to create an instance of the class that the method exists in
- Contains methods to create types of objects locally
- Use a string to create an instance of a type
- CreateInstance method to instantiate a class

Activator Class

- ```
  Type personType = typeof(Person);
  object obj = Activator.CreateInstance(personType);
  object[] personParams = new object[] { "JOHN SMITH", 222.0222f };
  personType.InvokeMember("ChangeValues", BindingFlags.InvokeMethod, null, obj,
  personParams);
  ```
  ```

Summary

- We've only just scratched the surface of what Reflection really is and all it's potential applications
- Allows programmers to inspect libraries that are unknown to them
- Reverse engineering of object hierarchy