Accessing Data from Program Code

Exercise 03 – Reflection and Advanced Programming

Agenda

- Recap – type definitions
- Generics
- Reflection
- Attributes
- Expression parsing (Visitor)

Refleciton

With examples in C#

Types

- A type defines the attributes and sometimes even the behavior of data (classes in OOP) used in the program
  - Static type system
    - Compiler checks for type definitions in compile time
  - Dynamic type system
    - Types are inferred in runtime
  - Difference between strictly typed and loosely typed – dynamic languages

Theoretical background and motivation

- Halting problem
  - Given a program, can we determine if the program will halt or loop forever?
  - The answer is NO, because system cannot reason about itself :)
  - Proof [not necessary]
    - Includes paradox (contradiction) that occurs when (inverted) program tries to evaluate if the target program will cause halt or infinite loop
      - Target program being the program itself
    - Reading the program definition in runtime seems familiar, right?

Reflection

How to read type definitions in the program runtime?

The answer is of course -> **REFLECTION!**

- Reflection can read the following definitions in runtime:
  - types
  - methods
  - properties
  - **attributes**
  - parameter names (if compiled with symbols)
- The information about the mentioned definitions are encapsulated in classes like `Type`, `MethodInfo`, `PropertyInfo`, `AttributeInfo`...

Examples (C#) - Type

- General information about the defined type can be inferred from `System.Type` class:
  - obj.GetType() -> returns the `Type` of the `obj`
  - `Type.FullName`
  - `Type.Namespace`
  - `Type.BaseType`
  - `Type.Assembly`
  - `Type.IsClass`
  - `Type.IsInterface`
  - `Type.IsEnum`
  - `Type.IsValueType`

Examples (C#) - Properties

- `type.GetProperties() -> PropertyInfo[]`
  - List all public properties
- `GetProperty(string name) -> PropertyInfo`
  - Get property by name
- `PropertyInfo.PropertyType -> Type`
  - Gets property type
- `PropertyInfo.GetValue(object obj) -> object?`
- `PropertyInfo.SetValue(object obj, object value)`

Examples (C#) - Constructor

- `GetConstructors() -> ConstructorInfo[]`
  - Lists all public constructors
- `GetConstructor(Type[]) -> ConstructorInfo`
  - Gets a specific constructor

Examples (C#) - Methods

- `type.GetMethods() -> MethodInfo[]`
  - Gets all public methods (instance + static)
- `GetMethod(string name) -> MethodInfo`
  - Gets a specific method by name
- `MethodInfo.GetParameters() -> ParameterInfo[]`
  - Gets method parameter info
- `MethodInfo.ReturnType`
  - Return type of a method
- `MethodInfo.Invoke(object obj, object?[]? parameters)`
  - Invokes method dynamically

Examples (C#) - Other

- `GetFields()  -> FieldInfo[]`
- `GetField(string) -> FieldInfo`
- `GetEvents() -> EventInfo[]`
- `GetInterfaces() -> Type[]`
- `GetNestedTypes() -> Type[]`

Examples (C#) – Binding flags

Flag	Description
BindingFlags.Public	Public members
BindingFlags.NonPublic	Private/protected/internal members
BindingFlags.Instance	Instance members
BindingFlags.Static	Static members
BindingFlags.FlattenHierarchy	Includes inherited static members

```
var privateFields = typeof(User).GetFields(
BindingFlags.NonPublic | BindingFlags.Instance);

foreach (var f in privateFields)
    Console.WriteLine(f.Name);
```

Limitations of reflection

- Reflection cannot be used to read the following definitions:
  - Generic (open type) parameter constraints
  - Comments (not transferred to class definitions in CLR)
  - region names (regions in C#, #region - #endregion)
  - type aliases (e.g., type of variable defined with `var` keyword in c#)

Attributes

With examples in C#

Attributes

- Meta data (extra behavior) added to compiled source code
- Can be applied to different units: class, method, property or assembly
- Instead of writing explicit code instructions, different definitions and behavior is hidden behind the attribute annotation
  - EquivaveInt is Java's **annotation** (e.g. `@Serializable`)

Examples...

- Already encountered previously when working with C#:
  - `[HttpGet]`
  - `[HttpPost]`
  - `[Serializable]`
  - `[Entity]`
  - `[Required]`
  - `[JsonObject]`
  - `[JsonProperty]`

User defined custom attributes

- New classes that inherit from base System.Attribute class
- `AttributeUsage` attribute can be used with extra flags to define targets
  - Using attributes to define your own attributes!
- This attributes define additional data for the specified target

Reading attributes

- Using reflection!

```
(MyAttribute) Attribute.GetCustomAttribute(
                                        type, typeof(MyAttribute))
```

Generics

- In languages that support generic types, it is possible to define classes and methods with type placeholders
  - `List<T>`
  - `JsonConvert.DeserializeObject<T>`
- Generics allow compile-time checks while maintaining flexibility

Source Code for Examples

- Will soon be available on github: https://github.com/bornaskracic/adpc-examples