C# Language Basic

Datatype
Control Flow
Iteration
Interface
Generics

Datatype

- Value Type
- Reference Type
- Value type stores its value directly, whereas a reference type stores a reference to the value.
- Value types are stored in an area known as the stack, and reference types are stored in an area known as the managed heap.

Value Types

int

System.Int32 | 32-bit | signed integer |-2,147,483,648 : 2,147,483,647 (-231:231–1)

float

System.Single | 32-bit | single-precision + 7 ±1.5 10245 to ±3.4 1038 floating point

double

System.Double | 64-bit | double-precision | 15/16 ±5.0 102324 to ±1.7 10308 floating point

char

System.Char | Represents a single 16-bit (Unicode) character

bool

System.Boolean | Represents true or false NA true false

Reference Type

object

System.Object | The root type. All other types in the CTS are derived (including value types) from object.

string

System.String |Unicode character string

Defining Constant Fields

- Defines variable with a fixed, unalterable value
- Once the value of a constant has been established, any attempt to alter it will result in a compile error
- The value assigned to constant variable must be known at compile time
- const fields are implicitly static

```
public const string EmployerName =“MS”

public const isEligibleForBouns = true;
```

Defining Read-Only Fields

- Allows you to establish a point of data whose values is not known at compile time but that should never change once established.
- Read only fields are not implicitly static like const fields

```
public readonly Tire = new Tire();

public readonly static Tire = new Tire();
```

Method Parameter Modifier

- (none)— Assumed to be pass by value
- out — Output parameters are assigned by the method being called(and therefore passed by reference)
- params — Allows you to send in variable number of identically typed arguments as a single logical parameter
- ref — Value is initially assigned by caller, and may be optionally reassigned by the called method.

Iteration Construct

- To repeat block of code until a terminating condition has been reached
  - for loop
  - foreach/in loop
  - while loop
  - do/while loop

Decision Constructs

- The if/else statement
- The switch statement

Interface

- An Interface is noting more than a named collection of semantically related abstract members.
- An Interface is defined using the C# “interface” keyword.
- Interface never specify a base class and contain members that do not take any access modifier (all are implicitly public)

```
 public interace ISomeInterface
 { bool SomeMember();}

public class SomeClass : ISomeInterface
{….}
```

Generics

Generics provide a way for programmer to define "placeholders" for method arguments and type definitions, which are specified at the time of invoking the generic method or creating the generic type

Thank You

@knitesh_