Enumeration and Structure Type

Enums

- An enumeration is a set of named integer constants.
- An enumerated type is declared using the **enum** keyword.
- Creates a list of compile-time constants
- Strongly typed
- Example:

```
public enum Color
{
   Red,
   Green,
   Blue
}
```

- Doc - https://msdn.microsoft.com/en-us/library/system.enum(v=vs.110).aspx

Declaring enum Types

- Declares a new type - derived from System.Enum, derived from ValueType
- Every enum type has a corresponding integral underlying-type
- default underlying type is **int**
- underlying type cannot be **char**
- **enum**s of different types require an explicit cast to convert between instances, even if the underlying type is the same
- Declaring an enum with a different underlying type:

```
public enum Color : ulong
{
   Red,
   Green,
   Blue
}
```

Enum members

- Each member must be named uniquely
- Each member has a value of the underlying type
- You may set values for the enum members
  - Different members may have the same value
- If you do not assign values, default values are assigned as follows:
  - If the first enum member has no assigned value, it defaults to 0
  - enum members after the first are assigned 1 + the value of the preceding member

```
public enum Color
{
   Red, // is assigned 0
   Green = 10,
   Blue // is assigned 11
}
```

Enum Members in an Enum

- You may refer to an enum member inside the enum declaration:

```
public enum Color
{
   Red,
   Green = 10,
   Blue,
   Max = Blue
}
```

- But, the reference may not result in a circular reference:

```
public enum Circular
{
   A = B, // this won’t work
   B
}
```

Using an enum Instance

```
enum Color
{
    Red,
    Blue,
    Green
}

class MyClass
{
    private Color color = Color.Red;

    private int GetIntFromColor()
    {
        return (int)color;
    }
    private string GetColorName()
    {
        return color.ToString();
    }
}
```

System.Enum

- Instance functions:
  - `string ToString()`
  - `int GetHashCode()`

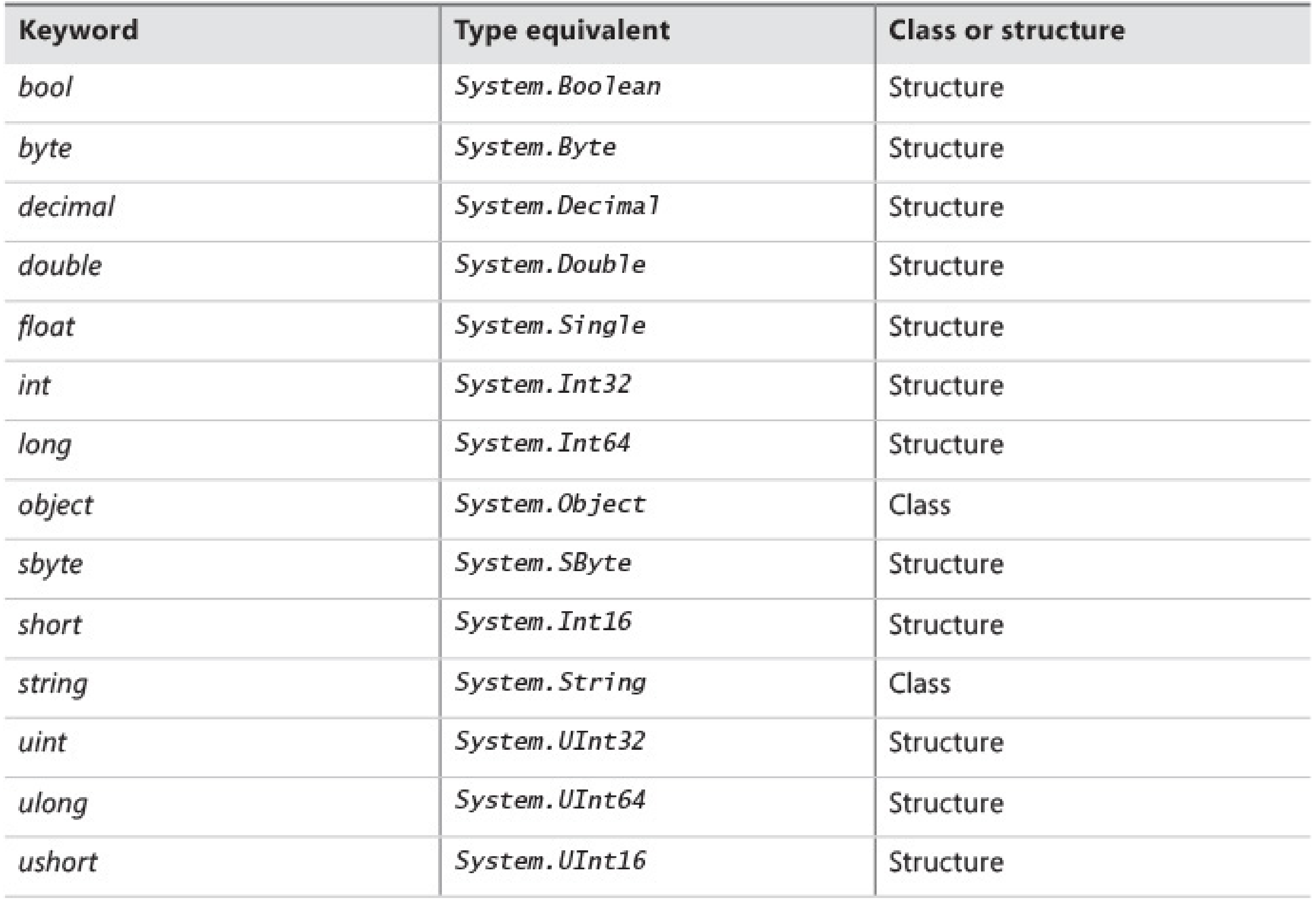
- Static functions:
  - `string GetName(Type enumtype, object value)`
  - `string[] GetNames(Type enumtype)`
  - `Type GetUnderlyingType(Type enumtype)`
  - `Array GetValues(Type enumtype)`
  - `bool IsDefined(Type enumtype, object value)`
  - `object Parse(Type enumtype, string value)`

Handy in helping convert the string to enumerated constants

Struct

- In case the Class contains so little data, it is better to define the type as a structure.
- A structure is a ***value*** type.
- Because structures are stored on the stack, as long as the structure is reasonably small, the memory management overhead is often reduced.
- Struct can have fields, methods and constructors
- The primitive numeric types (int, float etc) use struct.
- Doc - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-structs

Primitive Types Using Struct and Class



Keyword	Type equivalent	Class or structure
bool	*System.Boolean*	Structure
byte	*System.Byte*	Structure
decimal	*System.Decimal*	Structure
double	*System.Double*	Structure
float	*System.Single*	Structure
int	*System.Int32*	Structure
long	*System.Int64*	Structure
object	*System.Object*	Class
sbyte	*System.SByte*	Structure
short	*System.Int16*	Structure
string	*System.String*	Class
uint	*System.UInt32*	Structure
ulong	*System.UInt64*	Structure
ushort	*System.UInt16*	Structure

Declaring a struct

```
struct Date
   {
       private int year;
       private Month month;
       private int day;

       public Date(int ccyy, Month mm, int dd)
       {
           this.year = ccyy - 1900;
           this.month = mm;
           this.day = dd - 1;
       }

       public override string ToString()
       {
           string data = $"{this.month} {this.day + 1} {this.year + 1900}";
           return data;
       }

       public void AdvanceMonth()
       {
           this.month++;
           if(this.month == Month.December + 1)
           {
               this.month = Month.January;
               this.year++;
           }
       }
   }
```

Differences between Structures and Classes

- Struct can’t declare a default constructor

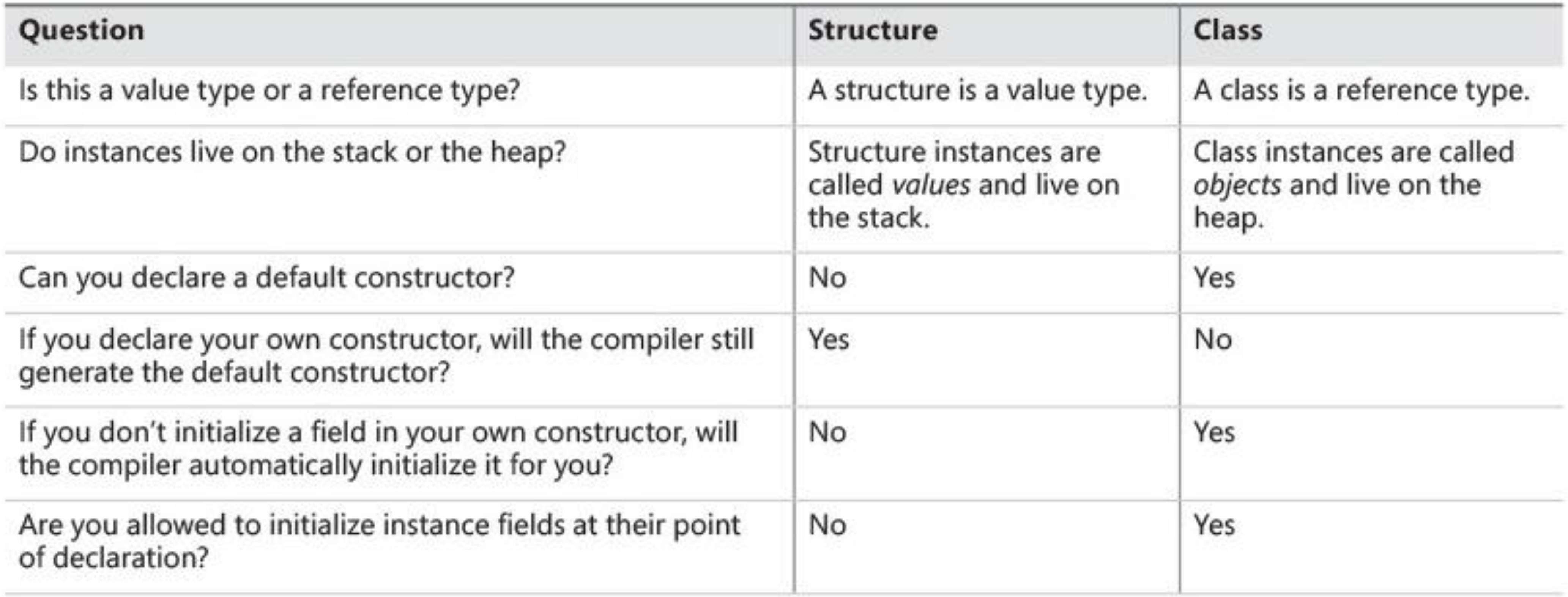
```
struct Time
{
    public Time() { ... } // compile-time error
    ...
}
```

- In a class, you can initialize instance fields at their point of declaration. In a structure, you cannot.

You can initialize struct members only by using a parameterized constructor or by accessing the members individually after the struct is declared. Any private or otherwise inaccessible members can be initialized only in a constructor.

```
struct Time
{
    private int hours = 0; // compile-time error
    private int minutes;
    private int seconds;
    ...
}
```

Differences between Structures and Classes



Question	Structure	Class
Is this a value type or a reference type?	A structure is a value type.	A class is a reference type.
Do instances live on the stack or the heap?	Structure instances are called *values* and live on the stack.	Class instances are called *objects* and live on the heap.
Can you declare a default constructor?	No	Yes
If you declare your own constructor, will the compiler still generate the default constructor?	Yes	No
If you don't initialize a field in your own constructor, will the compiler automatically initialize it for you?	No	Yes
Are you allowed to initialize instance fields at their point of declaration?	No	Yes

Some Struct Rules

Structs share most of the same syntax as classes, although structs are more limited than classes:

- Within a struct declaration, fields cannot be initialized unless they are declared as const or static.
- A struct cannot declare a default constructor (a constructor without parameters).
- Structs are copied on assignment. When a struct is assigned to a new variable, all the data is copied, and any modification to the new copy does not change the data for the original copy
- Structs are value types and classes are reference types.
- Unlike classes, structs can be instantiated without using a new operator.
- Structs can declare constructors that have parameters.
- A struct cannot inherit from another struct or class, and it cannot be the base of a class. All structs inherit
- directly from System.ValueType , which inherits from System.Object .
- A struct can implement interfaces.
- A struct can be used as a nullable type and can be assigned a null value.