Type conversion and type testing

Agenda

Boxing/unboxing

Implicit conversions

Explicit conversions

Convert methods

Anonymous types

The GetType method

typeof operator

is operator

Boxing/Unboxing

Convert value/reference types

Converting a value type to a reference type

The destination type must implement an interface implemented by the value type or be the object type

Boxing

The value type is wrapped inside a `System.Object` instance

The wrapped object is stored on the heap

Boxing

```
int i = 10;
object obj = i;
```

Unboxing

The reverse operation

Unboxing

```
object obj = 10;
int i = (int)obj;
```

Performance

Boxes are objects allocated on the heap

Too much boxing and unboxing can have a negative impact on the performance of your application.

Type conversion, implicit conversions

Convert types

It is possible to assign to a variable with a given type, a variable with a different type

How to convert

Implicit conversions

Explicit conversions

Helper classes

User defined conversions (not covered)

Convert types

A conversion can lead to a loss of data

A conversion can lead to a loss of precision

Implicit conversion

A type conversion that does not cause data loss is accepted by the compiler

No data loss

No error

No additional code to convert (assignment)

Loss of precision

Some numerical implicit conversions may cause a loss of precision (`int` to `float`...)

Implicit conversion

```
sbyte sbyte1 = 100;
short short1 = sbyte1;
```

Examples

Implicit numeric conversions

Implicit enumeration conversions

Implicit nullable conversions

Boxing conversions

A conversion from a type to its base type or implemented interface

Demo

Implicit conversion

Type conversion, explicit conversions

Explicit conversions

An explicit numeric conversion might result in data loss

Cast required

Cast expression

Converts explicitly a type to another

Compiling error if no explicit conversion

It is possible for a cast operation that compiles correctly to fail at run time (`InvalidCastException`)

Explicit conversion/Cast expression

```
short short1 = 200;
ulong ulong1 = (ulong)short1;
```

Examples

Implicit conversions

Explicit numeric conversions

Explicit enumeration conversions

Explicit nullable conversions

Unboxing conversions

A conversion from a base class to a derived class

As operator

Like casts, explicitly converts an expression to a reference type or a nullable value type

Unlike cast expressions, returns null and doesn’t throw an exception

Demo

Explicit conversion

Convert with helper methods

Convert with helper methods

`Convert` class

`Parse` methods

Convert class

Converts a base data type to another base data type

Supported types : `bool`, `char`, `string`, integer types, floating-point types

Can throw exceptions

Demo

Use the Convert class helper methods

Anonymous types

Anonymous types

Allows to create an object instance without creating a new type

It encapsulates read only properties

Anonymous types

The type of an anonymous type is a class that is created behind the scenes

The type has no name, it is anonymous

The name of this type is not accessible

Declaration

The type is inferred with `var`

Instantiated with the `new` keyword

Anonymous type

```
var movie = new { Id = 1, Title = "Title" };
```

Demo

Anonymous types

Type testing : the GetType method

Type testing

`Object.GetType`

`typeof` operator

`is` operator

Compile-time/run-time type

Compile-time type = type of the variable in the source code

The run-time type = the type of the instance of the variable

Object.GetType

Gets the runtime type of an instance

Return an implementation of the abstract **Type** class

Object.GetType()

```
int i = 1;
Type iType = i.GetType();
Console.WriteLine(iType);
// System.Int32
```

Demo

GetType method

Type testing : the typeof operator

typeof

Gets the Type instance of a type

Return an implementation of the abstract Type class

Demo

typeof operator

Type testing : the is operator

is

Returns true if an expression is compatible with a given type

Summary

Boxing is when you wrap a value type in an object or an object that implements an interface implemented by the value type

Unboxing is the reverse process

Boxing/unboxing can affect performance

An implicit conversion doesn’t need a cast, has no data loss, no error, but may lose precision

An explicit conversion needs a cast, may lose data and precision. Possible error (InvalidCastException)

The as operator converts but returns null instead of provoking an error

The Convert class convert a base data type to another

Summary

An anonymous type is an object created by encapsulating read-only properties

The object GetType method return the runtime type of an object. Return a Type instance

The typeof operator returns the Type instance associated with a given type

The is operator returns true if an expression is compatible with a given type

You can compare an object instance to null using the is operator