Type Conversion

Casting, boxing and unboxing

Canberra Institute of Technology

aie

SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

- Types, Variables, and Values
- Casting and Type Conversions
- Boxing and Unboxing
- dynamic and var

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Types, Variables, and Values

- C# is a strongly-typed language
  - Every variable and constant has a type
  - .NET defines built-in numeric types, and more complex types
- Information stored in a type can include:
  - Storage space required
  - Max and min value
  - Members (methods, fields, events)
  - Base type
  - Memory location for variables allocated at run-time
  - Kinds of permitted operations

Types, Variables, and Values

- Type info ensures all operations are *type safe*

```
int a = 5;
int b = a + 2; int //OK
```

Declare an `int`, the compiler allows you to perform addition and subtraction

```
bool test = true;
// Error. Operator '+' cannot be
// applied to operands of type
// 'int' and 'bool'.
int c = a + test;
```

The compiler knows you’re trying to do something stupid

Types, Variables, and Values

- After a variable is declared, it:
  - Cannot be re-declared with a new type
  - Cannot be assigned a value not compatible with its type
- Values can be converted to other types
- *Type conversions* not causing data loss are performed automatically
- Conversions causing data loss require a *cast*

Casting and Type Conversions

- Implicit
- Explicit (cast)
- User-defined
- Conversions with helper classes

Implicit Conversions

- No special syntax required
- No data lost
- For example:
  - Convert from smaller to larger integral type
  - Conversions from derived classes to base classes

```
// long can hold any value an int can hold, and more!
int num = 2147483647;
long bigNum = num;
```

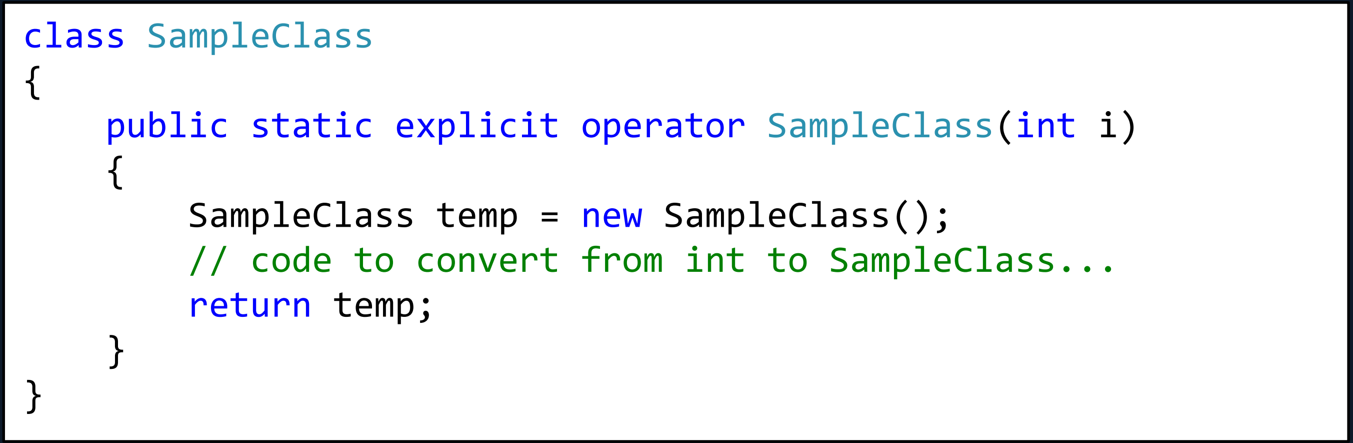
Explicit Conversion

- Require a cast operator
- Required when data might be lost, or when conversion might not succeed
- For example:
  - Numeric conversions to a type with less precision
  - From a base-class to a derived class

```
double x = 1234.7;
int a;
// Cast double to int.
a = (int)x;
```

User-Defined Conversions

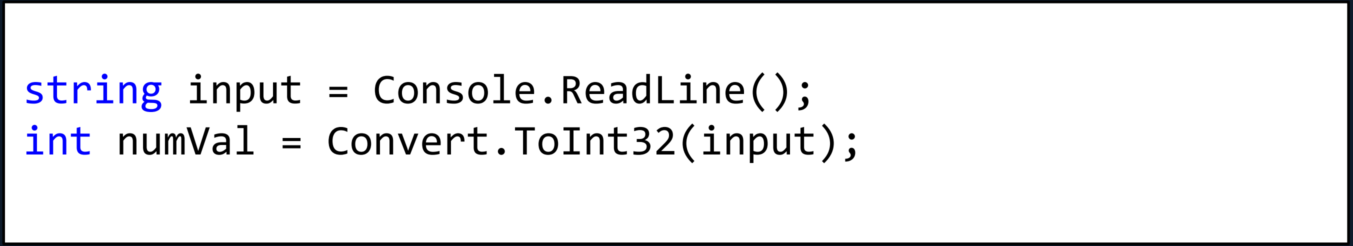
- Performed by special methods
- For types that do not have a base class-derived class relationship



```
class SampleClass
{
    public static explicit operator SampleClass(int i)
    {
        SampleClass temp = new SampleClass();
        // code to convert from int to SampleClass...
        return temp;
    }
}
```

Conversions with Helper Classes

- Convert between non-compatible types
- For example:
  - Convert between integers and System.DateTime objects
  - Convert a string to an int



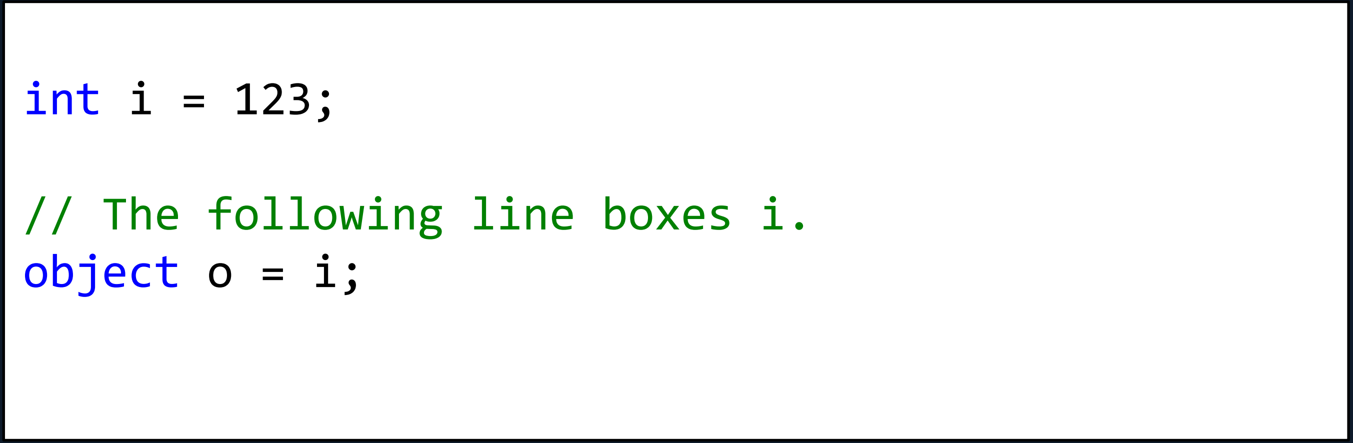
```
string input = Console.ReadLine();
int numVal = Convert.ToInt32(input);
```

Boxing and Unboxing

- In C#, a value of any type can be treated as an object
- *Boxing*
  - Convert a value type to the type *object*
  - Is implicit
  - Value is wrapped in a System.Object, stored on managed heap
- *Unboxing*
  - extracts the object
  - Is explicit

Boxing

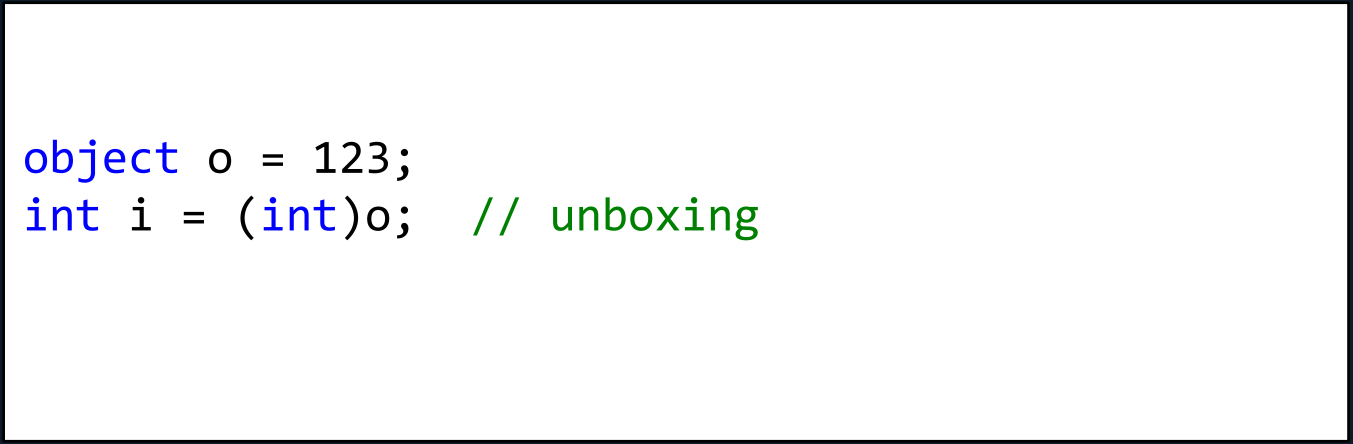
- the integer variable i is *boxed* and assigned to object o.



```
int i = 123;

// The following line boxes i.
object o = i;
```

Unboxing

- The object o can then be *unboxed* and assigned to integer variable i



```
object o = 123;
int i = (int)o;  // unboxing
```

Boxing and Unboxing

- Computationally expensive
  - When boxing, a new object must be allocated and constructed
  - The cast for unboxing is also expensive
- Used to store values on the garbage-collected heap
  - (otherwise the value is stored on the stack)

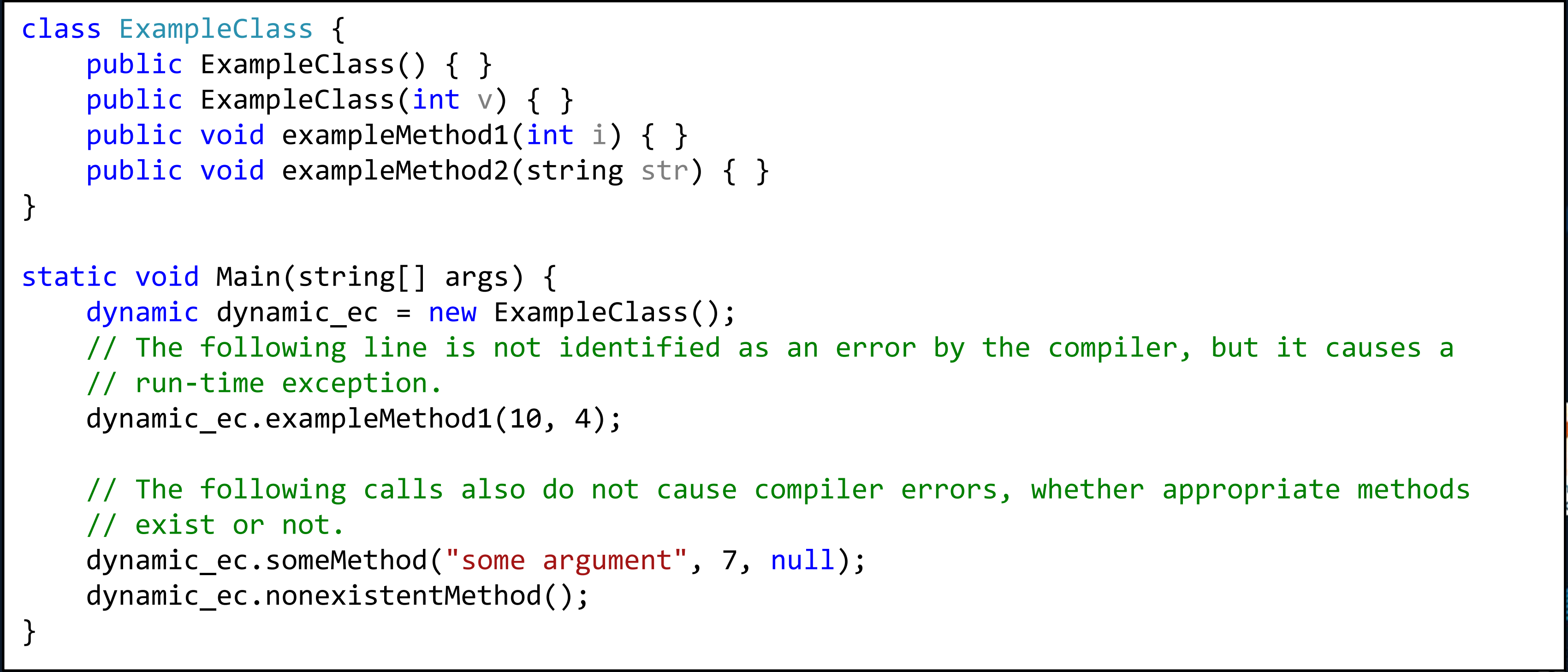
Boxing and Unboxing

- This used to be a big thing in .NET 1.1 and earlier
- Collection classes only worked with objects
- Now we have generic collection classes
- Might come up when working with older APIs
- Also comes up frequently when using *reflection*

dynamic

- C# 4.0 introduced the new type *dynamic*
- *dynamic* bypasses static type checking
- Errors are caught at run time
- Intellisense not available
- Facilitates interoperation with other (dynamic) languages and frameworks
  - In some cases might be easier and more convenient than reflection

dynamic

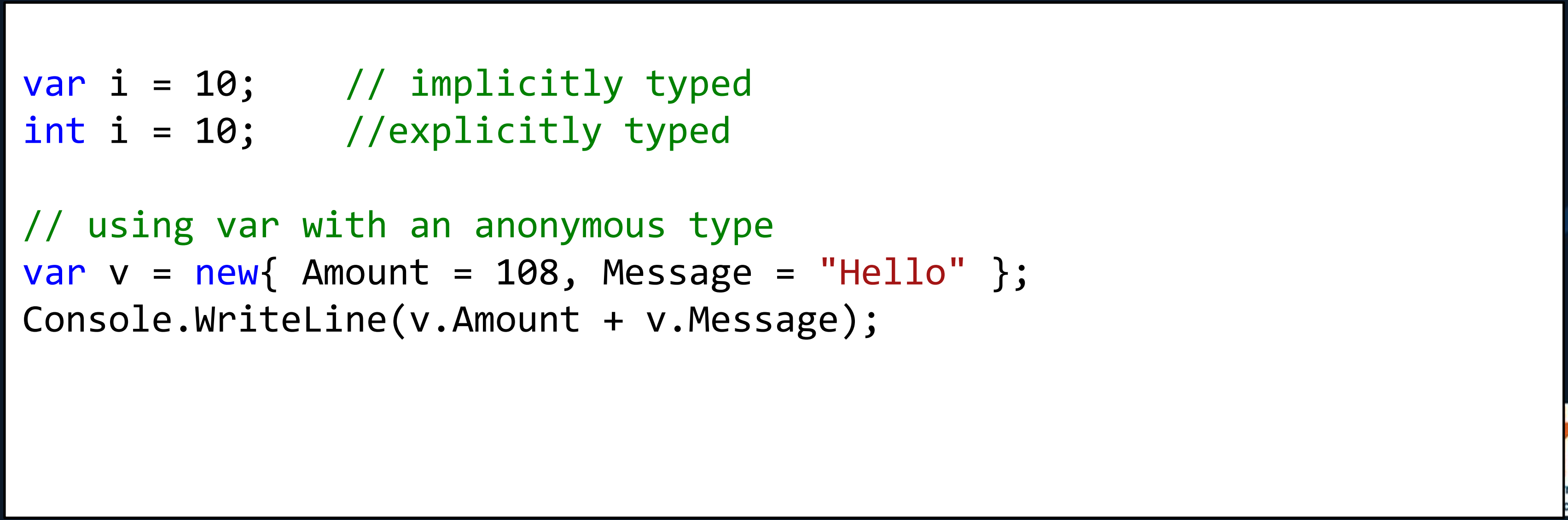


```
class ExampleClass {
    public ExampleClass() { }
    public ExampleClass(int v) { }
    public void exampleMethod1(int i) { }
    public void exampleMethod2(string str) { }
}

static void Main(string[] args) {
    dynamic dynamic_ec = new ExampleClass();
    // The following line is not identified as an error by the compiler, but it causes a
    // run-time exception.
    dynamic_ec.exampleMethod1(10, 4);

    // The following calls also do not cause compiler errors, whether appropriate methods
    // exist or not.
    dynamic_ec.someMethod("some argument", 7, null);
    dynamic_ec.nonexistentMethod();
}
```

var

- Introduced in C# 3.0
- Statically typed – type decided at compile time
- Errors caught at compile time
- Intellisense available
- Must initialize variables at time of declaration
- Mostly a choice of syntactic style
- Necessary when working with anonymous types

var



```
var i = 10;     // implicitly typed
int i = 10;     //explicitly typed

// using var with an anonymous type
var v = new{ Amount = 108, Message = "Hello" };
Console.WriteLine(v.Amount + v.Message);
```

Summary

- Implicit conversion requires no syntax, convert from smaller to larger numeric types
- Explicit conversions need a cast operator, used when data might be lost in the conversion
- User-defined casting uses special functions to convert between types
- Helper classes used to convert between non-compatible types
- Boxing/Unboxing converts between value types and objects
- dynamic and var and infer the type

References

- Microsoft MSDN. 2015. *Casting and Type Conversions*. [ONLINE] Available at:https://msdn.microsoft.com/en-us/library/ms173105.aspx. [Accessed 18 February 15].
- Microsoft MSDN. 2015. *Types*. [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/ms173104.aspx. [Accessed 18 February 15].
- Microsoft MSDN. 2015. *Boxing and Unboxing*. [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/yz2be5wk.aspx. [Accessed 18 February 15].
- Microsoft MSDN. 2015. *dynamic*. [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/dd264741.aspx. [Accessed 18 February 15].
- Alexandra Rusina. 2015. *Understanding the Dynamic Keyword in C# 4*. [ONLINE] Available at:https://msdn.microsoft.com/en-us/magazine/gg598922.aspx. [Accessed 18 February 15].
- Eric Lippert. 2014. *Uses and misuses of implicit typing*. [ONLINE] Available at:http://blogs.msdn.com/b/ericlippert/archive/2011/04/20/uses-and-misuses-of-implicit-typing.aspx. [Accessed 18 February 15].
- Microsoft MSDN. 2015. *Anonymous Types*. [ONLINE] Available at: https://msdn.microsoft.com/en-au/library/bb397696.aspx. [Accessed 18 February 15].