CHEAT SHEET

This cheat sheet is designed as a way for you to quickly study the key points of this chapter.

Conversion Basics

- Implicit conversion doesn't use a cast operator.
- Explicit conversion uses a cast operator.
- Widening conversions always succeed and a cast is optional. Magnitude is never lost but precision may be.
- Narrowing conversions do not always succeed, and a cast or other conversion method is required.
- Integer operations (including casting) that result in overflow or underflow are ignored unless you use a `checked` block or the Advanced Builds Settings dialog.
- Floating point operations that result in overflow or underflow are ignored. Check the result for the value `Infinity` or `NegativeInfinity` to see if overflow or underflow has occurred.
- You can cast arrays of references but be aware that the new array refers to the same array and not a new one.

The is and as Operators

- Use the `is` operator to determine if a variable is compatible with a certain type.
- Use the `as` operator to convert an object into a compatible type (or null if the object isn't compatible with the type).
- The `as` operator is particularly useful if you know an object's type, for example in an event handler.

Parsing

- Use the `Parse` method to parse text into a value. You must protect `Parse` method calls with `try-catch` blocks.
- Use the `TryParse` method to attempt to parse text and see if there is an error. `TryParse` returns `true` if it succeeds and `false` if there is an error.
- Use the `System.Globalization.NumberStyles` enumeration to allow `Parse` and `TryParse` to understand special symbols such as thousands separators, decimal points, and currency symbols.
- Some useful `NumberStyles` values include `Integer`, `HexNumber`, `Number`, `Float`, `Currency`, and `Any`.

Specialized Conversions

- The `System.Convert` class provides methods that convert from one data type to another.

- `System.Convert` methods include `ToBoolean`, `ToDouble`, `ToSingle`, `ToByte`, `ToInt16`, `ToString`, `ToChar`, `ToInt32`, `ToUInt16`, `ToDateTime`, `ToInt64`, `ToUInt32`, `ToDecimal`, `ToSByte`, and `ToUInt64`.
- The `System.BitConverter` class converts data to and from arrays of bytes.
- Boxing occurs when you convert a value type into a reference type as in `object obj = 72`. This is slow, so you should avoid it if possible.
- Unboxing occurs when you convert a reference type back into a value type.
- The `dynamic` type is a static type, but its value isn't evaluated until run time.

Strings

- `String`s are immutable.
- The intern pool holds an instance of every unique `String`.
- `StringBuilder`s are mutable and can be more efficient than `String`s for performing a long series of concatenations.
- The `StringWriter` and `StringReader` classes provide methods for writing and reading characters and lines with an underlying `StringBuilder` object.

Formatting

- The `ToString` and `String.Format` methods convert values into strings.
- `String.Format` uses composite format strings that can specify argument numbers, field widths, alignments, and format strings. Field indexes start at 0.
- Standard format strings are locale-aware so you should use them whenever possible.
- Useful standard numeric formatting strings include `C`/`c` (currency), `D`/`d` (decimal), `E`/`e` (exponential), `F`/`f` (fixed point), `G`/`g` (the shorter of `E` or `F`), `N`/`n` (number, as in `1,234.56`), `P`/`p` (percent), and `X`/`x` (hexadecimal).
- Useful standard `DateTime` formatting strings include `d` (short date), `D` (long date), `f` ("full" with short time), `F` ("full" with long time), `g` ("general" with short time), `G` ("general" with long time), `M` or `m` (month/day), `t` (short time), `T` (long time), and `Y` or `y` (year/month).

REVIEW OF KEY TERMS

boxing Boxing is the process of converting a value type such as `int` or `bool` into an object or an interface supported by the value's type. This enables a program to treat a simple value as if it were an object. See also unboxing.

Common Language Runtime (CLR) A virtual machine that manages execution of C# (and other .NET) programs.

composite format A format item used by `String.Format` to indicate how an argument should be formatted. The basic syntax is `{index[,length][:formatString]}`.

custom formatting string Enable you to build formats that are not provided by the standard formatting strings.

explicit conversion In an explicit conversion, the code uses an operator (such as a cast) or method (such as `int.Parse`) to explicitly tell the program how to convert a value from one type to another.

immutable A data type is immutable if its value cannot be changed after it has been created. The `String` class is immutable. `String` methods that seem to modify a `String`, such as `Replace` and `ToUpper`, actually replace the `String` with a new value containing the modified contents.

implicit conversion In an implicit conversion, the program automatically converts a value from one data type to another without any extra statements to tell it to make the conversion.

intern pool The CLR maintains a table called "intern pool" that contains a single reference to every unique string used by the program.

interoperability Interoperability enables managed code (such as a C# program) to use classes provided by unmanaged code that was not written under the control of the CLR.

narrowing conversion A narrowing conversion is a data type conversion where the destination type cannot hold every possible value provided by the source data type. Converting from a `long` to an `int` is a narrowing conversion because a `long` can hold values such as 4,000,000,000 that cannot fit in an `int`. Narrowing conversions must be explicit.

standard formatting string Enables you to determine how you want a value displayed at a high level.

unboxing Unboxing is the processing of converting a boxed value back into its original value type value. See also boxing.

Unicode Unicode is a standard for encoding characters used by scripts in various locales around the world. It enables a program to display English, Chinese, Kanji, Arabic, Cyrillic, and other character sets. The .NET Framework uses the UTF-16 encoding, which uses 16 bits to represent each character.

widening conversion A widening conversion is a data type conversion where the destination type can hold any value provided by the source data type; although, some loss of precision may occur. For example, converting from an `int` to a `long` is a widening conversion.

EXAM TIPS AND TRICKS

The Review of Key Terms and the Cheat Sheet for this chapter can be printed off to help you study. You can find these files in the ZIP file for this chapter at `www.wrox.com/remtitle.cgi?isbn=1118612094` on the Download Code tab.