## CHEAT SHEET

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

**Simple statements**

- Will end with a semicolon.
- They typically exist on one line but may extend to more than one line.
- Commonly used for variable declarations.
- They are also used for assignment statements.
- Typical usage scenarios are to perform a simple task.

**Complex statements**

- May or may not end with a semicolon with the `do-while` loop being an example of ending with a semicolon.
- Typically contain simple statements within the curly braces.
- Complex statements use curly braces to enclose other statements.
- They have structures, such as parentheses, that support their function.

**Boolean expressions**

- These are used in comparisons.
- C# uses the `bool` values `true` and `false` rather than `1` and `0` as in some other languages.
- They can be considered logical operators.
- They can exist in both unary and binary forms.
- Unary operates on a single operand, such as logical negation `!`.
- Binary operates on two operands, such as `&&` and `||`.

**if-then-else statements**

- Used for decision making.
- Will execute a code path depending on condition.
- Returns either true or false from the condition check.
- Doesn't require curly braces but use is recommended to help clarify what is included in the statement.
- Can be nested within other `if-then-else` statements.
- The `else` clause is used to choose alternative path for a `false` condition.
- The `else` if clause is used to choose alternative path for a `true` condition.

### switch statements

- Can check various data types in the condition in contrast to if statements.
- Uses case statements for each value to test against the condition.
- Switch statements are a cleaner code choice than nested if statements for code readability.
- Can use a default case when none of the cases return true.
- The break statement is used to end switch evaluation in a true case.
- They handle multiple conditions with a single set of instructions by removing the break from each case statement that holds the conditions to match.

### for statements

- Create a simple repetition structure.
- Uses an initialize component, a condition, and iterator in parentheses.
- Makes use of a statement block to contain one or more statements for execution enclosed in curly braces.
- Can be nested to create more complex looping structures.
- The initialization portion executes only at the start of the loop and not for each iteration.
- The condition portion is checked at each iteration.
- The increment portion happens only after the statements are executed in each iteration.
- This loop does not end with a semicolon.

### foreach statements

- They can be used for iterating over collections of items.
- They are best used when the number of values in collection is not known at design time.
- They work with any collection that implements `IEnumerable`.
- The declaration statement must use data types that are in the collection.

### while statements

- These execute similar to a `for` loop.
- The initialization is not part of the `while` loop; it takes place before the loop.
- The condition is evaluated at the start and on each iteration.
- The increment is accomplished within the loop.
- These are more intuitive than the `for` loop in terms of readability.

**do-while statements**

- Similar to the `while` loop, requires initialization outside of the loop structure.
- The condition is evaluated at the end of the loop.
- The increment is accomplished within the loop.
- The loop will execute at least once, regardless of condition.
- This loop style does end with a semicolon.

## REVIEW OF KEY TERMS

**assignment** Providing a value for a variable.

**Boolean** A value that is represented as either `true` or `false`.

**branching** Refers to changing code execution to a different path.

**condition** An evaluation of operands using logical operators.

**conditional instructions** Instructions that evaluate Boolean expressions and take action based on the outcome of the evaluation.

**comment** A code line that starts with the `//` characters and is a way of helping to document the code so that programmers can understand what the different code segments are intended to do.

**complex statement** A statement that can enclose one or more simple statements into a code block surrounded by curly braces `{}`. Typical complex statements are those used for repetition and decision structures such as `foreach()`, `if()`, `switch`, `do()` and so on.

**constant** A named value that is assigned at time of declaration and cannot be changed in code later.

**declaration** Used to create a variable in code.

**decrement** To decrease by a certain value.

**expression** An activity or code statement that returns a result.

**IEnumerable** A code component in C# that supports iteration.

**increment** To increase by a certain value.

**initialize** To set a starting value.

**iterator** A portion of loop that changes a value.

**literal** A notation used to indicate fixed values in code. Not the same as a constant. You cannot assign a value to a literal.

**loop** A repetition structure that repeats instructions.

**modulus** Remainder of integer division

**operator** Performs an operation on values.

**program flow** The logical execution of code.

**sentinel** A value used to signal the end for execution on a loop

**simple statement** A statement that ends with a semicolon and is typically used for program actions such as declaring variables, assigning values to variables, method calls, and code branching.

**spaghetti code** A term used to describe code that is complicated to follow and understand due to branching.

**statement** The code construct of the C# programming language that causes the application to perform an action.

**ternary operator** An operator that takes three arguments, a condition, a value for true, and a value for false.

**variables** Named values that can be changed in code.

## 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.