Appendix A

Answers to the Test Your Knowledge Questions

This appendix has the answers to the questions in the *Test your knowledge* section at the end of each chapter.

Chapter 1 – Hello, C#! Welcome, .NET!

Exercise 1.3 – Test your knowledge

Suggested answers to these questions:

1. Is Visual Studio better than VS Code?

   Answer: No. Each is optimized for different tasks. Visual Studio is large, heavyweight, and can create applications with graphical user interfaces, for example, Windows Forms, WPF, UWP, and .NET MAUI apps, but it is only available on Windows. Visual Studio is an **integrated development environment** (**IDE**) rather than a code editor. VS Code is smaller, more lightweight, code-focused, supports many more languages, and is available cross-platform.

2. Are .NET 5 and later versions better than .NET Framework?

   Answer: For modern development, yes, but it depends on what you need. .NET 5 and later are modern, cross-platform, performance-oriented versions of the legacy, mature .NET Framework. Modern .NET is more frequently updated. .NET Framework has better support for legacy applications; however, .NET Framework 4.8 will be the last release apart from security and bug fixes. It will never support some language features of C# 8 and later.

3. What is .NET Standard, and why is it still important?

   Answer: .NET Standard defines an API, aka contract, that a .NET platform can implement. The latest versions of .NET Framework, Xamarin, and modern .NET implement .NET Standard 2.0 to provide a single, standard API that developers can target for maximum reuse. .NET Core 3.0 and later implement .NET Standard 2.1, which has some new features not supported by .NET Framework. If you want to create a new class library that supports all .NET platforms, you will need it to be .NET Standard 2.0-compatible.

4. Why can a programmer use different languages (for example, C# and F#) to write applications that run on .NET?

   Answer: Multiple languages are supported on .NET because each one has a compiler that translates the source code into **intermediate language** (**IL**) code. This IL code is then compiled to native CPU instructions at runtime by the **Common Language Runtime** (**CLR**).

5. What is a top-level program, and how do you access any command-line arguments?

   Answer: A top-level program is a project that does not need to explicitly define a `Program` class with a `Main` method entry point, with a parameter named `args`, to access any command-line arguments. These are implicitly defined for you so that you can type statements without boilerplate code.

6. What is the name of the entry point method of a .NET console app, and how should it be explicitly declared if you are not using the top-level program feature?

   Answer: The entry point of a .NET console app is the `Main` method. An optional `string` array for command-line arguments and a return type of `int` are recommended, but they are not required. They can also be declared as asynchronous if the `await` keyword is used within the `Main` method.

   They can be explicitly declared, as shown in the following code:

   ```
   public static void Main() // Minimum
   public static int Main(string[] args) // Recommended

   public static async void Main() // Asynchronous
   public static async int Main(string[] args) // Asynchronous
   ```

   With .NET 6 and later, the entry point method of a console app is implicitly declared by the compiler using the top-level program feature, as shown in the following code:

   ```
   public static int <Main>$(String[] args)
   public static async int <Main>$(String[] args)
   ```

7. What namespace is the `Program` class defined in with a top-level program?

   Answer: With a top-level program, the `Program` class is defined in a `null` namespace.

8. Where would you look for help with a C# keyword?

   Answer: The Microsoft Learn website. Specifically, C# keywords are documented at the following link: `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/`.

9. Where would you look first for solutions to common programming problems?

   Answer: `https://stackoverflow.com/`.

10. What should you do after getting AI to write code for you?

    Answer: You should never trust the code written by AI, so test it thoroughly.

Chapter 2 – Speaking C#

Exercise 2.3 – Test your knowledge

Suggested answers to these questions:

1. What statement can you type in a C# file to discover the compiler and language version?

   Answer: `#error version`

2. What are the two types of comments in C#?

   Answer: The two types of comments in C# are a single-line comment, prefixed with `//`, and a multi-line comment, starting with `/*` and ending with `*/`. There are also XML comments, which are introduced in *Chapter 4*, *Writing, Debugging, and Testing Functions*, so I would not expect you to know about them after only reading *Chapter 2*.

3. What is the difference between a verbatim `string` and an interpolated `string`?

   Answer: A verbatim `string` is prefixed with the `@` symbol, and each character (except `"`) is interpreted as itself; for example, a backslash, `\`, is a backslash, `\`. An interpolated `string` is prefixed with the `$` symbol and can include expressions surrounded with braces, like this: `{expression}`.

4. Why should you be careful when using `float` and `double` values?

   Answer: You should be careful when using `float` and `double` values because they are not guaranteed to be accurate, especially when performing equality comparisons.

5. How can you determine how many bytes a type like `double` uses in memory?

   Answer: You can determine how many bytes a type such as `double` uses in memory by using the `sizeof()` operator, for example, `sizeof(double)`.

6. When should you use the `var` keyword?

   Answer: You should only use the `var` keyword to declare local variables when you cannot specify a known type. It is easy to overuse `var` due to its convenience when initially writing code, but its use can make it harder to maintain code later.

7. What is the newest syntax to create an instance of a class like `XmlDocument`?

   Answer: The newest way to create an instance of a class such as `XmlDocument` is to use a **target-typed new** expression, as shown in the following code:

   ```
   XmlDocument doc = new();
   ```

8. Why should you be careful when using the `dynamic` type?

   Answer: You should be careful when using the `dynamic` type because the type of object stored in it is not checked until runtime, which can mean runtime exceptions being thrown if you attempt to use a member that does not exist on the type.

9. How do you right-align a format `string`?

   Answer: To right-align a format `string`, after the index or expression, add a comma and an integer value to specify a column width within which to align the value. Positive integers mean right-aligned, and negative integers mean left-aligned.

10. What character separates arguments for a console app?

    Answer: The space character separates arguments for a console app.

Test your knowledge of number types

What type would you choose for the following "numbers"?

1. A person's telephone number

   Answer: `string`

2. A person's height

   Answer: `float` or `double`

3. A person's age

   Answer: `int` for best performance on most CPUs or `byte` (0 to 255) for the smallest size

4. A person's salary

   Answer: `decimal`

5. A book's ISBN

   Answer: `string`

6. A book's price

   Answer: `decimal`

7. A book's shipping weight

   Answer: `float` or `double`

8. A country's population

   Answer: `uint` (0 to about 4 billion)

9. The number of stars in the universe

   Answer: `ulong` (0 to about 18 quadrillion) or `System.Numerics.BigInteger` (which allows an arbitrarily large integer)

10. The number of employees in each of the small or medium businesses in the United Kingdom (up to about 50,000 employees per business)

    Answer: Since there are hundreds of thousands of small or medium businesses, we need to take memory size as the determining factor, so choose `ushort` because it only takes 2 bytes compared to `int`, which takes 4 bytes.

Chapter 3 – Controlling Flow, Converting Types, and Handling Exceptions

Exercise 3.2 – Practice exercises

Explore loops and overflow

What will happen if this code executes?

```
int max = 500;
for (byte i = 0; i < max; i++)
{
  WriteLine(i);
}
```

Answer:

The code will loop forever because the value of `i` can only be between `0` and `255`. Once `i` gets incremented beyond `255`, it loops back to `0` and, therefore, will always be less than `max` (`500`).

To prevent the infinite loop, you can add a checked statement around the code. This would cause an exception to be thrown after `255` due to the overflow, as shown in the following output:

```
254
255
System.OverflowException says Arithmetic operation resulted in an overflow.
```

A code solution can be found at the following link:

`https://github.com/markjprice/cs14net10/tree/main/code/Chapter03/Exercise_LoopsAndOverflow`.

Exercise 3.3 – Test your knowledge

Suggested answers to these questions:

1. What happens when you divide an `int` variable by `0`?

   Answer: `DivideByZeroException` is thrown when dividing an integer or decimal by `0`.

2. What happens when you divide a `double` variable by `0`?

   Answer: The `double` type contains a special value of `Infinity`. Instances of floating-point numbers can have the special values of `NaN` (**not a number**) or, in the case of dividing by `0`, either `PositiveInfinity` or `NegativeInfinity`. Those special values output as `8` and `-8`, which are supposed to look like infinity symbols on their sides (they are NOT eight and negative-eight!).

3. What happens when you overflow an `int` variable, that is, set it to a value beyond its range?

   Answer: It will loop unless you wrap the statement in a `checked` block, in which case, `OverflowException` will be thrown.

4. What is the difference between `x = y++;` and `x = ++y;`?

   Answer: In the statement `x = y++;`, the current value of `y` will be assigned to `x` and then `y` will be incremented, and in the statement `x = ++y;`, the value of `y` will be incremented and then the result will be assigned to `x`.

5. What is the difference between `break`, `continue`, and `return` when used inside a loop statement?

   Answer: The `break` statement will end the whole loop and continue executing after the loop. The `continue` statement will end the current iteration of the loop and continue executing at the start of the loop block for the next iteration, and the `return` statement will end the current method call and continue executing after the method call.

6. What are the three parts of a `for` statement, and which of them are required?

   Answer: The three parts of a `for` statement are the initializer, condition, and incrementer expressions. All three parts are optional. If the condition is missing, then it will loop forever unless you break out of the loop using `break`, `return`, or something else.

7. What is the difference between the = and == operators?

   Answer: The = operator is the assignment operator for assigning values to variables, while the == operator is the equality check operator that returns `true` or `false`.

8. Does the following statement compile – `for ( ; ; ) ;`?

   Answer: Yes. The `for` statement only requires the semicolons to separate the initializer expression, condition expression, and incrementor expressions. All three expressions are optional, so they can be missing. This `for` statement will execute the empty `;` statement after the closing brace, forever. It is an example of an infinite loop.

9. What does the underscore (_) represent in a `switch` expression?

   Answer: The underscore (_) represents the default return value. It is called a discard because it is used to represent a potential variable that is not needed. It can be used not just in `switch` expressions but also in other scenarios where a placeholder variable must be declared but its value is not needed, such as parameters for lambda expressions and tuple deconstruction.

10. What interface must an object "implement" to be enumerated over by using the `foreach` statement?

    Answer: An object must "implement" the `IEnumerable` interface. It must have the correct methods with the correct signatures, even if the object does not actually implement the interface.

Test your knowledge of operators

A code solution can be found at the following link:

`https://github.com/markjprice/cs14net10/tree/main/code/Chapter03/Exercise_Operators.`

1. What are the values of `x` and `y` after the following statements execute?

    ```
    x = 3;
    y = 2 + ++x;
    ```

    Answer: `x` is `4` and `y` is `6`.

2. What are the values of `x` and `y` after the following statements execute?

    ```
    x = 3 << 2;
    y = 10 >> 1;
    ```

    Answer: `x` is `12` and `y` is `5`.

3. What are the values of `x` and `y` after the following statements execute?

    ```
    x = 10 & 8;
    y = 10 | 7;
    ```

    Answer: `x` is `8` and `y` is `15`.

Chapter 4 – Writing, Debugging, and Testing Functions

Exercise 4.3 – Test your knowledge

Suggested answers to these questions:

1. What does the C# keyword `void` mean?

    Answer: It indicates that a method has no return value.

2. What are some differences between imperative and functional programming styles?

    Answer: An imperative programming style means writing a sequence of statements that the runtime executes step by step, like a recipe. Your code tells the runtime exactly how to perform the task. Do this. Now do that. It has variables, meaning that the state can change at any time, including outside the current function. Imperative programming causes side effects, changing the value of some state somewhere in your program. Side effects are tricky to debug. A functional programming style describes *what* you want to achieve instead of *how*. It can also be described as declarative. But the most important point is that functional programming languages make all states immutable by default to avoid side effects.

3. In VS Code or Visual Studio, what is the difference between pressing *F5*, *Ctrl* or *Cmd* + *F5*, *Shift* + *F5*, and *Ctrl* or *Cmd* + *Shift* + *F5*?

   Answer: *F5* saves, compiles, runs, and attaches the debugger; *Ctrl* or *Cmd* + *F5* saves, compiles, and runs the application without the debugger attached; *Shift* + *F5* stops the debugger and running application; and *Ctrl* or *Cmd* + *Shift* + *F5* restarts the application without the debugger attached.

4. Where does the `Trace.WriteLine` method write its output to?

   Answer: `Trace.WriteLine` writes its output to any configured trace listeners. By default, this includes the terminal or the command line, but can be configured to be a text file or any custom listener.

5. What are the five trace levels?

   Answer: 0 = Off, 1 = Error, 2 = Warning, 3 = Info, and 4 = Verbose.

6. What is the difference between the `Debug` and `Trace` classes?

   Answer: `Debug` is active only during development. `Trace` is active during development and after release into production.

7. When writing a unit test, what are the three "A"s?

   Answer: Arrange, Act, and Assert.

8. When writing a unit test using xUnit, what attribute must you decorate the test methods with?

   Answer: `[Fact]` or `[Theory]`.

9. What `dotnet` command executes xUnit tests?

   Answer: `dotnet test`.

10. What statement should you use to rethrow a caught exception named `ex` without losing the stack trace?

    Answer: Use `throw;`. Do not use `throw ex;` because this will lose stack trace information.

Chapter 5 – Building Your Own Types with Object-Oriented Programming

Exercise 5.2 – Practice with access modifiers

The code given in this exercise will cause the compiler to show the following errors:

```
Error CS0122 'Car' is inaccessible due to its protection level
Error CS0122 'Car.Start()' is inaccessible due to its protection level
Error CS0122 'Car.Wheels' is inaccessible due to its protection level
```

In the class library project, in `Car.cs`, you would need to apply the `public` access modifier to the class, as the default access modifier for a type is `internal`, and also to the two properties and the method, as the default access modifier for a member is `private`, as shown highlighted in the following code:

```
public class Car
{
  public int Wheels { get; set; }
  public public bool IsEV { get; set; }
  public void Start()
  {
    Console.WriteLine("Starting...");
  }
}
```

Exercise 5.3 – Test your knowledge

Suggested answers to these questions:

1. What are the seven type or member access modifier keywords and combinations of keywords, and what do they do?

   Answer: The seven combinations of access modifier keywords and their effects are described in the following list:

   - `private`: This modifier makes a type or member only visible inside the class.
   - `internal`: This modifier makes a type or member only visible inside the same assembly.
   - `protected`: This modifier makes a type or member only visible inside the class or derived classes.
   - `internal protected`: This modifier makes a type or member only visible inside the class, derived classes, or within the same assembly.
   - `private protected`: This modifier makes a type or member only visible inside the class or derived classes that are within the same assembly.
   - `public`: This modifier makes a type or member visible everywhere.
   - `file`: This modifier makes a type visible only within the same code file. It cannot be applied to a member.

2. What is the difference between the `static`, `const`, and `readonly` keywords when applied to a type member?

   Answer: The difference between the `static`, `const`, and `readonly` keywords when applied to a type member is described in the following list:

   - `static`: This keyword makes the member shared by all instances, and it must be accessed through the type, not an instance of the type.

- `const`: This keyword makes the field a fixed literal value that must never change because, during compilation, assemblies that use the field copy the literal value at the time of compilation.
- `readonly`: This keyword restricts the field so that it can only be assigned to using a constructor or field initializer at runtime.

3. What does a constructor do?

   Answer: A constructor allocates memory and initializes field values.

4. Why do you need to apply the `[Flags]` attribute to an `enum` type when you want to store combined values?

   Answer: If you don't apply the `[Flags]` attribute to an `enum` type when you want to store combined values, then a stored `enum` value that is a combination will be returned by a call to `ToString` as the stored integer value, instead of one or more of the comma-separated list of text values.

5. Why is the `partial` keyword useful?

   Answer: You can use the `partial` keyword to split the definition of a type over multiple files. It is commonly used with source-generated code.

6. What is a tuple?

   Answer: A tuple is a data structure consisting of multiple parts. It is used when you want to store multiple values as a unit without defining a type for them.

7. What does the `record` keyword do?

   Answer: The `record` keyword defines a data structure that is immutable by default to enable a more functional programming style. Like a class, a record can have properties and methods, but the values of properties can only be set during initialization.

8. What does overloading mean?

   Answer: Overloading is when you define more than one method with the same method name and different input parameters.

9. What is the difference between the following two statements? (Do not just say a `>` character!)

   ```
   public List<Person> Children = new();
   public List<Person> Children => new();
   ```

   Answer: The first statement defines a field named `Children` and initializes it to an empty list of `Person` objects. The second statement defines a read-only property named `Children` that returns an empty list of `Person` objects. A field is a data storage location that can be referenced. A property is one or a pair of methods that get and/or set a value. The value of a property is often stored in a `private` field. A read-only property only has a `get` method that can be implemented using lambda expression syntax.

10. How do you make a method parameter optional?

    Answer: You make a method parameter optional by assigning a default value to it in the method signature.

11. Why can't a `DateTime` field be a constant?

    Answer: A `DateTime` field cannot be declared as a constant because `const` in C# is restricted only to compile-time constants of the built-in primitive types and a few specific types. `DateTime` is a struct, not a primitive constant type. While it's a value type and immutable, it's still too complex to be evaluated entirely at compile time. A `DateTime` field is constructed, for example, via `new DateTime(2025, 1, 1)`, and the constructor involves method calls, which are not allowed in constant expressions.

Chapter 6 – Implementing Interfaces and Inheriting Classes

Exercise 6.3 – Test your knowledge

Suggested answers to these questions:

1. What is a delegate?

   Answer: A delegate is a type-safe method reference. It can be used to execute any method with a matching signature.

2. What is an event?

   Answer: An event is a field that is a delegate with the `event` keyword applied. The keyword ensures that only `+=` and `-=` are used; this safely combines multiple delegates without replacing any existing event handlers.

3. How are a base class and a derived class related, and how can the derived class access the base class?

   Answer: A derived class (or subclass) is a class that inherits from a base class (or superclass). Inside a derived class, you use the `base` keyword to access the class that the subclass inherits from.

4. What is the difference between `is` and `as` operators?

   Answer: The `is` operator returns `true` if an object can be cast to a type; otherwise, it returns `false`. The `as` operator returns a reference to the object if an object can be cast to the type; otherwise, it returns `null`.

5. Which keyword is used to prevent a class from being derived from or a method from being further overridden?

   Answer: `sealed`.

6. Which keyword is used to prevent a class from being instantiated with the `new` keyword?

   Answer: `abstract`.

7. Which keyword is used to allow a member to be overridden?

   Answer: `virtual`.

8. What's the difference between a destructor and a deconstruct method?

   Answer: A destructor, also known as a finalizer, must be used to release resources owned by an object. A deconstruct method is a feature of C# 7 or later that allows a complex object to be broken down into smaller parts. It is especially useful when working with tuples.

9. What are the signatures of the constructors that all exceptions should have?

   Answer: The signatures of the three constructors that all exceptions should have are shown in the following list:

   - A constructor with no parameters
   - A constructor with a `string` parameter, usually named `message`
   - A constructor with a `string` parameter, usually named `message`, and an `Exception` parameter, usually named `innerException`

10. What is an extension method, and how do you define one?

    Answer: An extension method is a compiler trick that makes a `static` method of a `static` class appear to be one of the members of another type. You define which type you want to extend by prefixing the first parameter of that type in the method with the `this` keyword.

Chapter 7 – Packaging and Distributing .NET Types

Exercise 7.3 – Test your knowledge

Suggested answers to these questions:

1. What is the difference between a namespace and an assembly?

   Answer: A namespace is the logical container of a type. An assembly is the physical container of a type. To use a type, a developer must reference its assembly. Optionally, the developer can import its namespace or specify the namespace when specifying the type.

2. How do you reference another project in a `.csproj` file?

   Answer: You reference another project in a `.csproj` file by adding a `<ProjectReference>` element that sets its `Include` attribute on a path to the reference project file inside an `<ItemGroup>` element, as shown in the following markup:

   ```
   <ItemGroup>
     <ProjectReference Include="..\Calculator\Calculator.csproj" />
   </ItemGroup>
   ```

3. What is the benefit of a tool such as ILSpy?

   Answer: A benefit of a tool such as ILSpy is learning how to write code in C# for the .NET platform by seeing how other packages are written. You should never steal their intellectual property, of course. However, it is especially useful to see how Microsoft developers have implemented key components of the base class libraries. Decompiling can also be useful when calling a third-party library, and you need to better understand how it works to call it appropriately.

4. Which .NET type does the C# `float` alias represent?

   Answer: `System.Single`.

5. What are the `<PropertyGroup>` and `<ItemGroup>` elements used for in a .NET project file?

   Answer: The `PropertyGroup` element is used to define configuration properties that control the build process. These properties can include anything, from the output path of the compiled binaries to versioning information.

   The `ItemGroup` element is used to define collections of items that are included in the build process. Items can be source files, references to other assemblies, package references, and other resources.

6. What is the difference between framework-dependent and self-contained deployments of .NET applications?

   Answer: Framework-dependent modern .NET applications require .NET to be installed for an operating system to execute. Self-contained .NET applications include everything necessary to execute on their own.

7. What is a RID?

   Answer: **RID** is the acronym for **Runtime Identifier**. RID values are used to identify target platforms where a .NET application runs.

8. What is the difference between the `dotnet pack` and `dotnet publish` commands?

   Answer: The `dotnet pack` command creates a NuGet package that could then be uploaded to a NuGet feed, such as Microsoft's. The `dotnet publish` command puts the application and its dependencies into a folder for deployment to a hosting system.

9. How can you control the sources that a build process uses to download NuGet packages?

   Answer: A `nuget.config` file allows you to specify, prioritize, and configure NuGet package sources, as shown in the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="nuget.org"
         value="https://api.nuget.org/v3/index.json" />
    <add key="MyCustomSource"
```

```
          value="https://mycustomsource.com/nuget" />
  </packageSources>
</configuration>
```

10. What are some limitations of using AOT compilation?

    Answer: Native AOT has limitations that apply to the projects that enable it, some of which are shown in the following list:

    - No dynamic loading of assemblies. For example, you cannot use `Assembly.LoadFrom()`.
    - No runtime code generation, for example, when using `System.Reflection.Emit`.
    - It requires trimming.
    - The projects must be self-contained, so they must embed any libraries they call, which increases their size.

Chapter 8 – Working with Common .NET Types

Exercise 8.3 – Test your knowledge

Suggested answers to these questions:

1. What is the maximum number of characters that can be stored in a `string` variable?

   Answer: The maximum size of a `string` variable is 2 GB, or about 1 billion characters, because each character uses 2 bytes due to the internal use of Unicode (UTF-16) encoding for characters in a `string`.

2. When and why should you use a `SecureString` type?

   Answer: The string type leaves text data in the memory for too long, and it's also too visible. The `SecureString` type encrypts its text and ensures that the memory is released immediately. For example, in **Windows Presentation Foundation** (**WPF**), the `PasswordBox` control stores its password as a `SecureString` variable, and when starting a new process, the `Password` parameter must be a `SecureString` variable.

3. When is it appropriate to use a `StringBuilder` class?

   Answer: When concatenating more than three `string` variables, you will use less memory and get improved performance using `StringBuilder` than using the `string.Concat` method or the + operator.

4. When should you use a `LinkedList<T>` class?

   Answer: Each item in a linked list has a reference to its previous and next siblings, as well as the list itself. A linked list should be used when items need to be inserted and removed from positions in the list without moving the items in memory.

5. When should you use a `SortedDictionary<T>` class rather than a `SortedList<T>` class?

Answer: The `SortedList<T>` class uses less memory than `SortedDictionary<T>`, but `SortedDictionary<T>` has faster insertion and removal operations for unsorted data. If a list can be populated all at once from sorted data, `SortedList<T>` will be constructed faster than `SortedDictionary<T>`.

6. In a regular expression, what does `$` mean?

   Answer: In a regular expression, `$` represents the end of the input.

7. In a regular expression, how can you represent digits?

   Answer: In a regular expression, you can represent digit characters using `\d` or `[0-9]`.

8. Why should you *not* use the official standard for email addresses to create a regular expression to validate a user's email address?

   Answer: The effort is not worth the pain for you or your users. Validating an email address using the official specification doesn't check whether that address exists, or whether the person entering the address is its owner.

9. What characters are output when the following code runs?

```
string city = "Aberdeen";
ReadOnlySpan<char> citySpan = city.AsSpan()[^5..^0];
WriteLine(citySpan.ToString());
```

   Answer: `rdeen`. `^5..` means the range is 5 characters long. `..^0` means the range ends zero characters in from the right end.

10. How could you check that a web service is available before calling it?

    Answer: Use the `Ping` class to call the web service and check the `Status` property of the reply.

Chapter 9 – Working with Files, Streams, and Serialization

Exercise 9.3 – Test your knowledge

Suggested answers to these questions:

1. What is the difference between using the `File` class and the `FileInfo` class?

   Answer: The `File` class has `static` methods and cannot be instantiated. It is best used for one-off tasks such as copying a file. The `FileInfo` class requires the instantiation of an object that represents a file. It is best used when you need to perform multiple operations on the same file.

2. What is the difference between the `ReadByte` method and the `Read` method of a stream?

   Answer: The `ReadByte` method returns a single byte each time it is called, and the `Read` method fills a temporary array with bytes up to a specified length. It is generally best to use `Read` to process multiple bytes at once.

3. When would you use the `StringReader`, `TextReader`, and `StreamReader` classes?

   Answer:

   You would use the classes as described in the following list:

   - `StringReader` is used to efficiently read from a string stored in memory
   - `TextReader` is an abstract class that `StringReader` and `StreamReader` both inherit from for their shared functionality
   - `StreamReader` is used to read strings from a stream that can be any type of text file, including XML and JSON

4. What does the `DeflateStream` type do?

   Answer: `DeflateStream` implements the same compression algorithm as `gzip` but without a cyclical redundancy check; therefore, although it produces smaller compressed files, it cannot perform integrity checks when decompressing.

5. How many bytes per character does UTF-8 encoding use?

   Answer: The number of bytes per character used by the UTF-8 encoding depends on the character. Most Western alphabet characters are stored using one byte. Other characters may need two or more bytes.

6. What is an object graph?

   Answer: An object graph is any set of connected instances of classes that reference each other. For example, a `Customer` object may have a property named `Orders` that references a collection of `Order` instances.

7. What is the best serialization format to choose to minimize space requirements?

   Answer: **JavaScript Object Notation** (**JSON**) has a good balance between space requirements and practical factors such as human readability, but the **protocol buffers** (**Protobuf**) serialization format used by the gRPC standard is best for minimizing space requirements.

8. What is the best serialization format to choose for cross-platform compatibility?

   Answer: There is still an argument for **eXtensible Markup Language** (**XML**) if you need maximum compatibility, especially with legacy systems, although JSON is better if you need to integrate with web systems, and Protobuf offers the best performance and minimum bandwidth use.

9. Why is it bad to use a `string` value such as `"\Code\Chapter01"` to represent a path, and what should you do instead?

   Answer: It is bad to use a `string` value such as `"\Code\Chapter01"` to represent a path because it assumes that backslashes are used as a folder separator on all operating systems. Instead, you should use the `Path.Combine` method and pass separate `string` values for each folder, or a `string` array, as shown in the following code:

   ```
   string path = Path.Combine(new[] { "Code", "Chapter01" });
   ```

10. Where can you find information about NuGet packages and their dependencies?

    Answer: You can find information about NuGet packages and their dependencies at the following link: `https://www.nuget.org/`.

Chapter 10 – Working with Data Using Entity Framework Core

Exercise 10.3 – Test your knowledge

Suggested answers to these questions:

1. What type would you use for the property that represents a table, for example, the `Products` property of a database context?

   Answer: `DbSet<T>`, where `T` is the entity type, for example, `Product`.

2. What type would you use for the property that represents a one-to-many relationship, for example, the `Products` property of a `Category` entity?

   Answer: `ICollection<T>`, where `T` is the entity type, for example, `Product`.

3. What is the EF Core convention for primary keys?

   Answer: The property named `ID` or `Id` or `ClassNameID` or `ClassNameId` is assumed to be the primary key. If the type of that property is any of the following, then the property is also marked as being an `IDENTITY` column: `tinyint`, `smallint`, `int`, `bigint`, or `guid`.

4. When might you use an annotation attribute in an entity class?

   Answer: You might use an annotation attribute in an entity class when the conventions cannot work out the correct mapping between the classes and tables, for example, if a class name does not match a table name or a property name does not match a column name. You might also define constraints, such as a maximum length of characters in a text value or a range of numeric values, by decorating with validation attributes. These can be read by technologies such as ASP.NET Core MVC and Blazor to provide automatic validation warnings to users.

5. Why might you choose Fluent API in preference to annotation attributes?

   Answer: You might choose Fluent API in preference to annotation attributes when you want to keep your entity classes free from extraneous code that is not needed in all scenarios. For example, when creating a .NET Standard 2.0 class library for entity classes, you might want to only use validation attributes so that the metadata can be read by EF Core, as well as by technologies such as ASP.NET Core model binding validation and .NET MAUI desktop and mobile apps. However, you might want to use Fluent API to define EF Core-specific functionality, such as mapping to a different table or column name.

6. What does a transaction isolation level of `Serializable` mean?

   Answer: Maximum locks are applied to ensure complete isolation from any other processes working with the affected data.

7. What does the `DbContext.SaveChanges` method return?

   Answer: An `int` value for the number of entities affected.

8. What is the difference between eager loading and explicit loading?

   Answer: Eager loading means related entities are included in the original query to the database so that they do not have to be loaded later. Explicit loading means related entities are not included in the original query to the database, and they must be explicitly loaded just before they are needed.

9. How should you define an EF Core entity class to match the following table?

   ```
   CREATE TABLE Employees(
     EmpId INT IDENTITY,
     FirstName NVARCHAR(40) NOT NULL,
     Salary MONEY
   )
   ```

   Answer: Use the following class:

   ```
   public class Employee
   {
     [Column("EmpId")]
     public int EmployeeId { get; set; }

     [Required]
     [StringLength(40)]
     public string FirstName { get; set; }

     [Column(TypeName = "money")]
     public decimal? Salary { get; set; }
   }
   ```

10. What benefit do you get from declaring entity navigation properties as `virtual`?

    Answer: You can enable lazy loading if you declare entity navigation properties as `virtual`.

Chapter 11 – Querying and Manipulating Data Using LINQ

Exercise 11.3 – Test your knowledge

Suggested answers to these questions:

1. What are the two required parts of LINQ?

   Answer: A LINQ provider and the LINQ extension methods. You must import the `System.Linq` namespace to make the LINQ extension methods available, and you must reference a LINQ provider assembly for the type of data that you want to work with.

2. Which LINQ extension method would you use to return a subset of properties from a type?

   Answer: The `Select` method allows the projection (aka selection) of properties.

3. Which LINQ extension method would you use to filter a sequence?

   Answer: The `Where` method allows filtering by supplying a delegate (or lambda expression) that returns a Boolean to indicate whether the value should be included in the results.

4. List five LINQ extension methods that perform aggregation.

   Answer: Any five of the following: `Max`, `Min`, `Count`, `LongCount`, `Average`, `Sum`, and `Aggregate`.

5. What is the difference between the `Select` and `SelectMany` extension methods?

   Answer: `Select` returns exactly what you specify to return. `SelectMany` checks that the items you have selected are themselves `IEnumerable<T>` and then breaks them down into smaller parts. For example, if the type you select is a `string` value (which is `IEnumerable<char>`), `SelectMany` will break each `string` value returned into its individual `char` values and combine them into a single sequence.

6. What is the difference between `IEnumerable<T>` and `IQueryable<T>`? How do you switch between them?

   Answer: The `IEnumerable<T>` interface indicates a LINQ provider that will execute the query locally, such as LINQ to Objects. These providers have no limitations, but can be less efficient. The `IQueryable<T>` interface indicates a LINQ provider that first builds an expression tree to represent the query and then converts it into another query syntax before executing it, such as how EF Core converts LINQ into SQL statements. These providers sometimes have limitations, such as a lack of support for some expressions, and may throw exceptions. You can convert from an `IQueryable<T>` provider to an `IEnumerable<T>` provider by calling the `AsEnumerable` method.

7. What does the last type parameter, `T`, in generic `Func` delegates, such as `Func<T1, T2, T>`, represent?

   Answer: The last `T` type parameter in generic `Func` delegates such as `Func<T1, T2, T>` represents the type of the return value. For example, for `Func<string, int, bool>`, the delegate or lambda function used must return a Boolean value.

8. What is the benefit of a LINQ extension method that ends with `OrDefault`?

   Answer: The benefit of a LINQ extension method that ends with `OrDefault` is that it returns the default value, instead of throwing an exception if it cannot return a value. For example, calling the `First` method on a sequence of `int` values would throw an exception if the collection is empty, but the `FirstOrDefault` method would return `0`.

9. Why is query comprehension syntax optional?

   Answer: Query comprehension syntax is optional because it is just syntactic sugar. It makes code easier for humans to read, but it does not add any additional functionality except the `let` keyword.

   You can learn more about the `let` keyword at the following link: `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/let-clause`.

10. How can you create your own LINQ extension methods?

    Answer: Create a `static` class with a `static` method, with an `IEnumerable<T>` parameter prefixed with `this`, as shown in the following code:

```
namespace System.Linq
{
  public static class MyLinqExtensionMethods
  {
    public static IEnumerable<T> MyChainableExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // return something IEnumerable<T>
    }

    public static int? MyAggregateExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // return some int value
    }
  }
}
```

Chapter 12 – Introducing Modern Web Development Using .NET

Exercise 12.3 – Test your knowledge

Suggested answers to these questions:

1. What was the name of Microsoft's first dynamic server-side-executed web page technology, and why is it still useful to know this history today?

   Answer: **Active Server Pages** (**ASP**). The name ASP.NET Core derives from ASP, and it is still used in new features such as Tag Helpers, as shown in the following markup:

   ```
   <a asp-controller="Home" asp-action="Index">Home</a>
   ```

2. What are the names of two Microsoft web servers?

   Answer: Kestrel (cross-platform) and **Internet Information Services** (**IIS**), which is Windows-only.

3. What are some differences between a microservice and a nanoservice?

   Answer: A microservice implements more than one function grouped into a small domain and is always running. A nanoservice implements a single function and is not always running (i.e., it can be "serverless").

4. What is Blazor?

   Answer: Blazor is a .NET-based technology for implementing web user interfaces. It is designed to be used instead of **single-page applications** (**SPAs**) such as React, Angular, and Vue.

5. What was the first version of ASP.NET Core that could not be hosted on .NET Framework?

   Answer: ASP.NET Core 3.0 requires .NET Standard 2.1, which is not supported by .NET Framework.

6. What is a user agent?

   Answer: A user agent is a client to a web server, for example, a web browser or a search engine web crawler.

7. What impact does the HTTP request-response communication model have on web developers?

   Answer: Website dynamic code resides on and executes on a web server. The server code cannot trigger communication. A web browser must make an HTTP request to trigger code on the server, which can then generate an HTTP response. This makes updating a web page difficult because it is under the control of the client, not the server, when requests for more data are made.

8. Name and describe four components of a URL.

   Answer: The *scheme* determines whether you are using HTTP or HTTPS. The *domain* is the unique address of the computer (or a web farm of servers acting as a single logical computer). The *port number* is the port on which the server(s) listen (usually `80` for HTTP and `443` for HTTPS). The *path* is the relative path to a resource, such as a folder or file. The *query string* is for optional parameters passed along with the request. The *fragment* is an identified element within a resource, such as a web page.

9. What capabilities does Developer Tools give you?

   Answer: Developer Tools allows you to see every HTTP request and response, allows you to view client-side application data such as cookies, sessions, and local storage, provides a console for logging, and so on.

10. What are the three main client-side web development technologies, and what do they do?

    Answer: HTML5 (structure), CSS3 (styles), and JavaScript (execute actions).

Know your webbreviations

What do the following web abbreviations stand for, and what do they do?

1. **URI** (**Uniform Resource Identifier**) is a unique sequence of characters that identifies a resource.
2. **URL** (**Uniform Resource Locator**) is the address of a unique resource.
3. **WCF** (**Windows Communication Foundation**) is a framework for building service-oriented applications. It is part of .NET Framework, and an open source implementation named WCF Core is available for modern .NET.
4. **TLD** (**Top-Level Domain**) is one of the domains at the highest level in the hierarchical **DNS** (**Domain Name System**) of the internet, for example, `packt.com`.
5. **API** (**Application Programming Interface**) is a mechanism for a computer system to allow other computer systems to communicate with it. APIs should be well-documented.
6. **SPA** (**Single-Page Application**) is a web app that loads only a single web page and then updates the content dynamically via JavaScript APIs when needed. SPA frameworks include Angular, React, Vue, and Blazor.
7. **CMS** (**Content Management System**) is a software application that allows users to build and manage a website without having to write code.
8. **Wasm** (**WebAssembly**) is a binary instruction format and is designed as a compilation target for programming languages such as C#, for deployment on the web, client, and server.
9. **SASS** (**Syntactically Awesome Style Sheets**) is a CSS preprocessor. SASS has features that don't exist in CSS yet, such as nesting, mixins, and inheritance, that help you write maintainable CSS.
10. **REST** (**REpresentational State Transfer**) is an architectural style for distributed hypermedia systems. Roy Fielding presented it in 2000 in his famous dissertation.

In case you missed it (ICYMI): Acronyms and initialisms are types of abbreviation. Learn more at the following link: `https://www.rd.com/article/acronym-vs-abbreviation-whats-the-difference/`.

Chapter 13 – Building Websites Using ASP.NET Core

Exercise 13.3 – Test your knowledge

Suggested answers to these questions:

1. List six method names that can be specified in an HTTP request.

   Answer: `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, and `DELETE`. Others include `TRACE`, `OPTIONS`, and `CONNECT`.

2. List six status codes and descriptions that can be returned in an HTTP response.

   Answer: `200` OK, `201` Created, `301` Moved Permanently, `400` Bad Request, `404` Not Found (missing resource), and `500` Internal Server Error. Others include `101` Switching Protocols (e.g., from HTTP to WebSocket), `202` Accepted, `204` No Content, `304` Not Modified, `401` Unauthorized, `403` Forbidden, `406` Not Acceptable (for example, requesting a response format that is not supported by a website), and `503` Service Unavailable.

3. In ASP.NET Core, what is the `Program` class used for?

   Answer: In ASP.NET Core, the `Program` class is where you add and configure dependency services such as Razor Pages, MVC, and EF Core data contexts. It is also where you configure middleware in the request and response pipeline. This might include error handling, security options, static files, default files, and endpoint routing.

4. What does the acronym HSTS stand for, and what does it do?

   Answer: **HSTS** stands for **HTTP Strict Transport Security**, which is an opt-in security enhancement. If a website specifies it and a browser supports it, then it forces all communication over HTTPS and prevents a visitor from using untrusted or invalid certificates.

5. How do you enable static HTML pages for a website?

   Answer: To enable static HTML pages for a website, you must add statements to use default files and then static files (this order is important!), as shown in the following code:

   ```
   app.UseDefaultFiles(); // index.html, default.html, and so on

   // .NET 8 and earlier
   app.UseStaticFiles();

   // .NET 9 and later
   app.MapStaticAssets();
   ```

6. How do you mix C# code into the middle of HTML to create a dynamic page?

   Answer: To mix C# code into the middle of HTML to create a dynamic page, you can create a Blazor component file with the `.razor` file extension, or a Razor View or Razor Page file with the `.cshtml` file extension, and then prefix any C# expressions with the `@` symbol. C# statements need to be wrapped in braces, as shown in the following markup:

```
@{
  string[] DaysOfTheWeek =
    System.Threading.Thread.CurrentThread
    .CurrentCulture.DateTimeFormat.DayNames;

  string WhatDayIsIt =
    System.DateTime.Now.ToString("dddd");
}
<html>
  <head>
    <title>Today is @WhatDayIsIt</title>
  </head>
  <body>
    <h1>Days of the week in your culture</h1>
    <ul>
    @{
      foreach (string dayName in DaysOfTheWeek)
      {
        <li>@dayName</li>
      }
    }
    </ul>
  </body>
</html>
```

7. How can you define shared layouts for Blazor components?

   Answer: To define shared layouts for Blazor components, create a `MainLayout.razor` file that will define the markup for the shared layout.

8. In a Blazor project, what three files do you typically create in a `Components` folder, and what do they do?

   Answer: In the `Components` folder, you typically create the following three `.razor` files:

   - `_Imports.razor`: This file imports namespaces for all `.razor` files so that you do not need to import them at the top of every `.razor` file. At a minimum, you will want to import the namespace for Blazor routing and your local project's Blazor components.

- `App.razor`: This file contains HTML for the web page that will contain all your Blazor components. It also needs to reference your Blazor routes component somewhere in the `<body>` element of the web page.
- `Routes.razor`: This file defines a `<Router>` component that scans the current assembly for page components and their registered routes.

9. How do you configure an EF Core data context for use with an ASP.NET Core website?

   Answer: To configure an EF Core data context for use with an ASP.NET Core website, follow these steps:

   - In the project file, reference the assembly that defines the data context class.
   - In `Program.cs` or the `Startup` class, import the namespaces for `Microsoft.EntityFrameworkCore` and the data context class.
   - In the `ConfigureServices` method or the section of `Program.cs` that configures services, add a statement that configures the data context with a database connection string for use with a specified database provider, such as SQLite or SQL Server, as shown in the following code:

     ```
     services.AddDbContext<MyDataContext>(options =>
     // or UseSqlServer()
       options.UseSqlite("my database connection string"));
     ```

   - In the Blazor component's `@inject` section, declare a field to store the data context, as shown in the following code:

     ```
     @inject MyDataContext db;
     ```

10. What are the tasks that you must complete to enable Blazor and its static SSR capability in an existing ASP.NET Core Empty project?

    Answer: Create a `Components` folder to contain your Blazor components and a `Components\Pages` folder to contain your Blazor page components. In the `Components` folder, create three `.razor` files: `_Imports.razor`, `App.razor`, and `Routes.razor`. In `Components\Pages`, create an `Index.razor` file, a Blazor page component that will be your home page, shown by default by the Blazor router. It will need a directive at the top of the file to define a route for the root path: `@page "/"`. In `Program.cs`, you must call `AddRazorComponents()` to register Blazor, aka Razor components (`*.razor`) files, with ASP.NET Core's dependency services collection, and then call `MapRazorComponents<App>()` to map endpoints for all the found Blazor routes. You must also call `UseAntiforgery()` because Blazor components automatically check for anti-forgery tokens; therefore, the HTTP pipeline must enable middleware to support them.

Chapter 14 – Building Interactive Components Using Blazor

Exercise 14.3 – Test your knowledge

Suggested answers to these questions:

1. What are the four Blazor render modes, and how are they different?

   Answer: The four Blazor render modes are static SSR, streaming, interactive server, and interactive WebAssembly:

   - **SSR**: Executes code on the server side as Razor Pages and MVC do. The complete response is sent to the browser for display to the visitor.
   - **Streaming rendering**: Executes code on the server side. HTML markup can be displayed in the browser, and while the connection is still open, more markup is sent by the server to update the contents of the page. This improves the experience for the visitor because they see some content while waiting for the rest.
   - **Interactive server rendering**: Executes code on the server side, which means that code has full and easy access to server-side resources like databases. This can simplify implementing functionality. Interactive requests are made using SignalR, which is more efficient than a full request. A permanent connection is needed between the browser and server, which limits scalability.
   - **Interactive WebAssembly rendering**: Executes code on the client side, which means the code only has access to resources within the browser. This can complicate the implementation because a callback to the server must be made whenever new data is required.

2. In a Blazor Web App project, compared to an empty ASP.NET Core project, what extra configuration is required?

   Answer: In the section that configures services, you must call `AddRazorComponents`. To use interactive server rendering, you must also call `AddServerComponents`. To use interactive client rendering, you must also call `AddWebAssemblyComponents`. In the section that configures the HTTP pipeline, you must call `MapRazorComponents<App>` when setting up endpoint mapping to all the components configured in the `App.razor` file.

3. Why should you avoid the Blazor Server and Blazor Server Empty project templates?

   Answer: The Blazor Server and Blazor Server Empty project templates are now legacy, since they create projects where all the components are rendered on the server and interactions are handled by SignalR. Use the Blazor Web App project template instead, and then enable interactive server rendering in the components that need it.

4. In a Blazor Web App project, what do the `App.razor` and `Routes.razor` files do?

   Answer: The `App.razor` file is the root Razor component for your Blazor application. Think of it as the bootstrapper: it provides a HTML page template with `<head>` and `<body>` elements, wires up static assets, routing, client interactivity, and decides what to render when a user navigates to a URL. Every request in your Blazor app ultimately flows through `App.razor`.

   The `Routes.razor` file is the router definition, the bit that knows how to match URLs to your Razor components and apply layouts. It scans your app assembly for Razor components decorated with `@page`, and handles matching the current URL to the right page component. If a Blazor page component doesn't specify a layout, it wraps it in `MainLayout` by default.

   To summarize, `App.razor` is the outer HTML document and `Routes.razor` is the router host that decides which page and layout to render.

5. What is the main benefit of using the `<NavLink>` component?

   Answer: The main benefit of using the `<NavLink>` component is that it integrates with the Blazor routing system, so it can automatically apply a current style to visually indicate when the current route matches the `<NavLink>` component.

6. How can you pass a value into a component?

   Answer: You can pass a value into a component by decorating a `public` property in the component with the `[Parameter]` attribute and then setting the attribute in the component when using it, as shown in the following code:

```
// defining the component
@code {
  [Parameter]
  public string ButtonText { get; set; };
}

// using the component
<CustomerDetail ButtonText="Create Customer" />
```

7. What is the main benefit of using the `<EditForm>` component?

   Answer: The main benefit of using the `<EditForm>` component is automatic validation messages.

8. How can you execute some statements when parameters are set?

   Answer: You can execute some statements when parameters are set by defining an `OnParametersSet` method or an `OnParametersSetAsync` method.

9. How can you execute some statements when a component appears?

   Answer: You can execute some statements when a component appears by defining an `OnInitialized` method or an `OnInitializedAsync` method.

10. One of the benefits of Blazor is being able to implement client-side components using C# and .NET instead of JavaScript. Does a Blazor component need any JavaScript?

    Answer: Yes, Blazor components need some minimal JavaScript. For Blazor Server, this is provided by the `_framework/blazor.server.js` file. For Blazor WebAssembly, this is provided by the `_framework/blazor.webassembly.js` file. Blazor WebAssembly, with a **progressive web application** (PWA), also uses a JavaScript service worker file, `service-worker.js`. JavaScript is also needed to invoke browser and other client-side APIs, such as getting the current geolocation or interacting with browser storage and alert dialog boxes.

Chapter 15 – Building and Consuming Web Services

Exercise 15.3 – Test your knowledge

Suggested answers to these questions:

1. ASP.NET Core has multiple project templates for building web services. What are they, and how do you create them using the CLI?

   Answer: ASP.NET Core has three project templates to build web services with the following commands and switches:

   - **ASP.NET Core Web API** (using Minimal API): .NET 8 and later: `dotnet new webapi`. .NET 7 and earlier: `dotnet new webapi --use-minimal-apis`.
   - **ASP.NET Core Web API** (using controllers): .NET 8 and later: `dotnet new webapi --use-controllers`. .NET 7 and earlier: `dotnet new webapi`.
   - **ASP.NET Core Web API (native AOT)**: .NET 8 and later: `dotnet new webapiaot`.

2. When configuring an HTTP client, how do you specify the format of data that you prefer in the response from the web service?

   Answer: When configuring an HTTP client, you specify the format of the response that you prefer by adding an `Accept` header to the HTTP request that specifies the document format you prefer, and if you specify multiple formats and want to give them different weights, then set quality values between `0.0` and `1.0`, as shown highlighted in the following code:

```
builder.Services.AddHttpClient(name: "Northwind.WebApi",
  configureClient: options =>
  {
    options.BaseAddress = new Uri("https://localhost:5002/");
    options.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue(
      mediaType: "application/json", quality: 1.0));
  });
```

3. Why did the ASP.NET Core team replace the Swashbuckle package with their own implementation of OpenAPI support?

   Answer: In .NET 8 and earlier, the third-party Swashbuckle package was used to provide OpenAPI documentation, but the Swashbuckle project is no longer actively maintained by its community owner. Issues have not been addressed or resolved, and there is no official release for .NET 8. With .NET 9, the dependency on `Swashbuckle.AspnetCore` has been removed from the project template, and the capabilities of `Microsoft.AspNetCore.OpenApi` have been extended to provide OpenAPI document generation.

4. What must you do to specify what responses should be expected when calling `MapGet` or a similar method?

   Answer: To specify what responses should be expected when calling `MapGet` or a similar method, you need to use metadata methods such as: `Produces` / `Produces<T>`, `ProducesResponseType`, and `Accepts`. These methods attach metadata to your endpoint that tools like OpenAPI, analyzers, and clients can use to know exactly what responses to expect, as shown in the following code:

   ```
   app.MapGet("/weather", () =>
   {
       return Results.Ok(new WeatherForecast("Sunny", 25));
   })
   .Produces<WeatherForecast>(StatusCodes.Status200OK)
   .Produces(StatusCodes.Status404NotFound);
   ```

5. List three methods that can be called to return responses with different status codes.

   Answer: Three methods that can be called to return responses with different status codes include the following:

   - `Ok`: This returns the `200` status code and the object passed to this method in the body.
   - `CreatedAtRoute`: This returns the `201` status code and the object passed to this method in the body.
   - `NoContentResult`: This returns the `204` status code and an empty body.
   - `BadRequest`: This returns the `400` status code and an optional error message.
   - `NotFound`: This returns the `404` status code and an optional error message.

6. List four ways that you can test a web service.

   Answer: Four ways that you can test a web service include the following:

   - Using a browser to test simple HTTP `GET` requests
   - Installing the REST Client extension for VS Code
   - Installing the Swagger NuGet package in your web service project, enabling Swagger, and then using the Swagger testing user interface
   - Installing the Postman tool from the following link: `https://www.postman.com`

I listed the four preceding ways because they are techniques that I covered or mentioned in the chapter. There are, of course, many other ways to test a web service, for example, cURL. Give yourself an extra point for any valid methods beyond the ones that I have listed.

7. Why should you not wrap your use of `HttpClient` in a `using` statement to dispose of it when you are finished, even though it implements the `IDisposable` interface, and what should you use instead?

   Answer: `HttpClient` is shared, reentrant, and partially thread-safe, so it is tricky to use correctly in many scenarios. You should use `HttpClientFactory`, which was introduced in .NET Core 2.1.

8. What are the benefits of HTTP/2 and HTTP/3 compared to HTTP/1.1?

   Answer: HTTP/2 benefits from full multiplexing, which reduces latency, supports request prioritization, and minimizes overhead in the protocol using header compression. HTTP/3 benefits from being based on UDP rather than TCP, so that any packet loss does not block all streams. HTTP/3 also has 0-RTT support, meaning subsequent connections do not repeat the TLS acknowledgment, so the client can start requesting data faster.

9. How can you enable clients to detect whether your web service is healthy with ASP.NET Core 2.2 and later?

   Answer: To enable clients to detect whether your web service is healthy, you can install health check APIs, including database health checks for EF Core data contexts. Health checks can be extended to report detailed information back to the client.

10. What are the main types of object caching, and why is `HybridCache` the best?

    Answer: There are three types of object caching – in-memory, distributed, and hybrid caching:

    - In-memory caching stores data in the memory of the web server where the application is running. This is useful for small to medium-sized applications, where the caching needs are not too extensive and can be handled by a single server's memory.
    - Distributed caching allows you to cache data across multiple servers, making it suitable for large-scale, distributed applications. This ensures data availability and consistency across different nodes in a web farm.
    - The HybridCache API addresses some limitations found in the `IDistributedCache` and `IMemoryCache` APIs. It provides a single interface for both in-process and out-of-process caching. `HybridCache` can seamlessly replace any existing `IDistributedCache` and `IMemoryCache` usage. It always uses the in-memory cache initially, and when an `IDistributedCache` implementation is available, `HybridCache` leverages it for secondary caching. This dual-level caching approach combines the speed of in-memory caching with the durability of distributed or persistent caching. `HybridCache` prevents cache stampedes, which occur when a frequently used cache entry is invalidated, causing multiple requests to try to repopulate it simultaneously. `HybridCache` merges concurrent operations, ensuring that all requests for the same response wait for the first request to be completed.