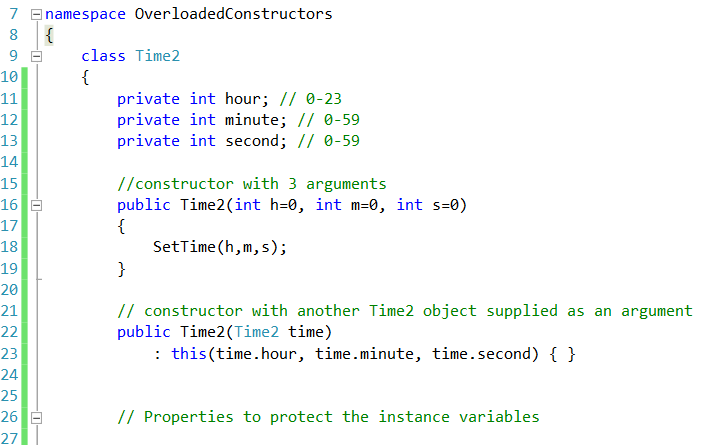
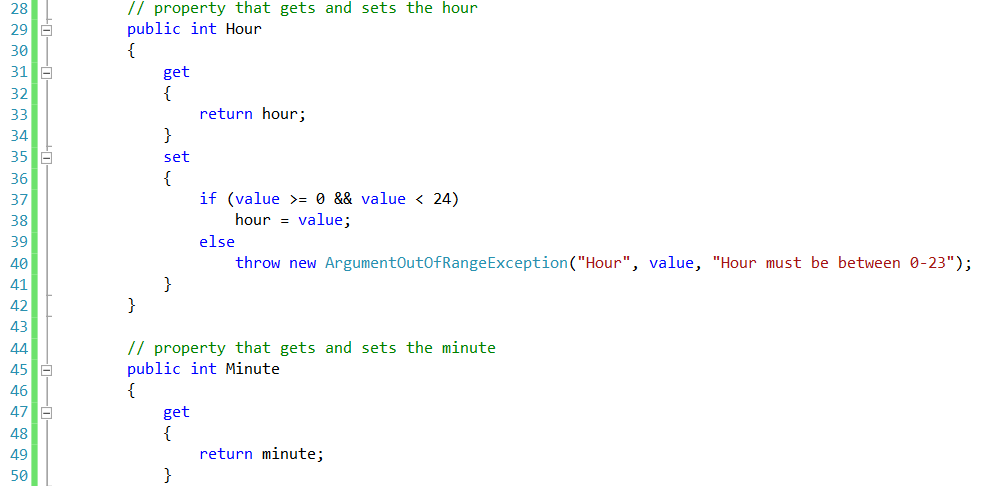
Systems Development: Object Oriented Programming

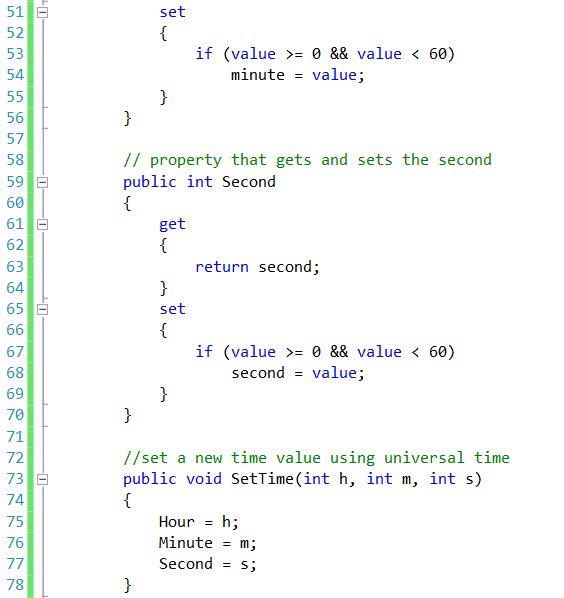
(H172 35)

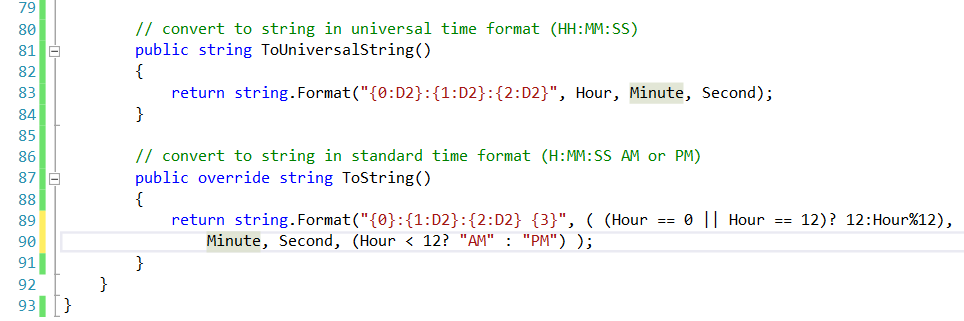
Overloaded Constructors Walkthrough

Overloaded constructors enable objects of that class to be conveniently initialised in different ways. To overload constructors, simply provide multiple constructor declarations with different signatures









In this class we have 2 constructors for Time2. The compiler invokes the appropriate Time2 constructor by matching the number and types of the arguments specified in the constructor call with the number and types of the parameters specified in each constructor declaration.

Class Time2’s Parameterless Constructor

Lines 16–19 declare a constructor with three default parameters. This constructor is also considered to be the class’s **parameterless** constructor because you can call the constructor without arguments and the compiler will automatically provide the default parameter values i.e. hour, minute, second will be initialised to 0. This constructor can also be called with one argument for the hour (minute and second will automatically be initialised to 0), two arguments for the hour and minute (second will automatically be initialised to 0), or three arguments for the hour, minute and second. This constructor calls SetTime to set the time.

Class Time2’s Constructor That Receives a Reference to Another Time2 Object

Lines 22–23 declare a Time2 constructor that receives as an argument a reference to a Time2 object. In this case, the values from the Time2 argument are passed to the three-parameter constructor (one constructor invokes the other constructor) at lines 16–19 to initialize the hour, minute and second. In this constructor, we use this in a manner that’s allowed only in the constructor’s header. In line 22, the usual constructor header is followed by a colon (:), then the keyword this. The this reference is used in method-call syntax (along with the three int arguments) to invoke the Time2 constructor that takes three int arguments (lines 16–19). The constructor passes the values of the time argument’s Hour, Minute and Second properties to set the hour, minute and second of the Time2 object being constructed. Additional initialization code can be placed in this constructor’s body (this example has an empty body) and it will execute after the other constructor is called.

Constructor Initializers

The use of the *this* reference as shown in line 23 is called a constructor initializer. Constructor initializers are a popular way to reuse initialization code provided by one of the class’s constructors rather than defining similar code in another constructor’s body. This syntax makes the class easier to maintain, because one constructor reuses the other. If we needed to change how objects of class Time2 are initialized, only the constructor at lines 16–19 would need to be modified. Even that constructor might not need modification— it simply calls the SetTime method to perform the actual initialization, so it’s possible that the changes the class might require would be localized to this method.

Line 23 could have directly accessed instance variables hour, minute and second of the constructor’s time argument with the expressions time.hour, time.minute and time.second—even though they’re declared as private variables of class Time2, but instead we are using the properties.

Class Time2’s SetTime Method

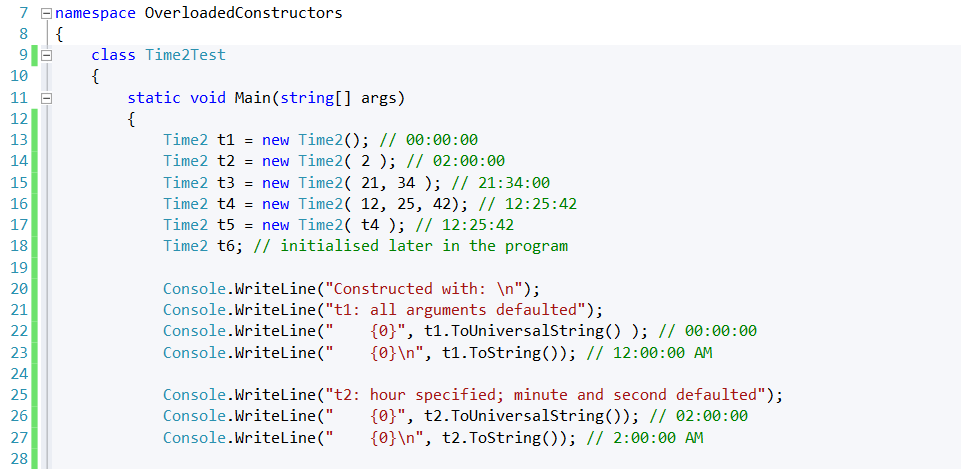
Method SetTime (lines 73–78) invokes the set accessors of the new properties Hour (lines 29–42), Minute (lines 45–56) and Second (lines 59–70), which ensure that the value supplied for hour is in the range 0 to 23 and that the values for minute and second are each in the range 0 to 59. If a value is out of range, each set accessor throws an ArgumentOutOfRangeException. In this example, we use the Argument-OutOfRangeException constructor that receives three arguments—the name of the item that was out of range, the value that was supplied for that item and an error message.

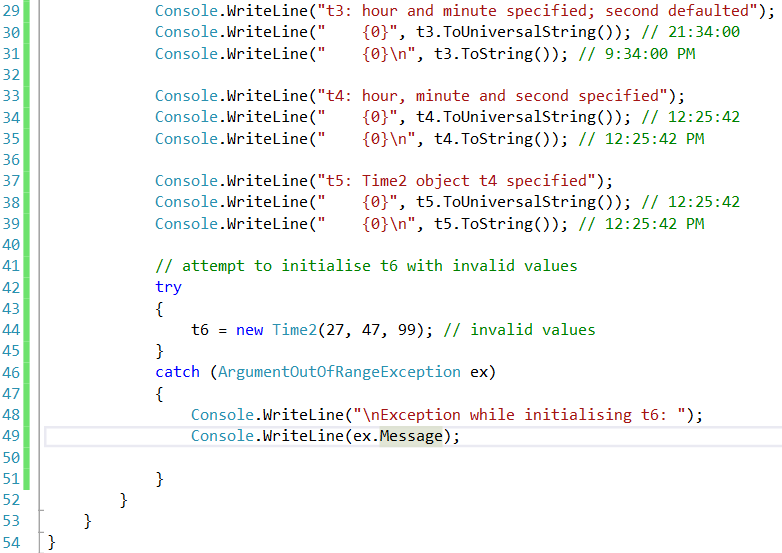
Notes Regarding Class Time2’s Methods, Properties and Constructors

Time2’s properties are accessed throughout the class’s body. Method SetTime assigns values to properties Hour, Minute and Second in lines 75–77, and methods ToUniversalString and ToString use properties Hour, Minute and Second in line 83 and lines 89–90, respectively. These methods could have accessed the class’s private data directly. However, consider changing the representation of the time from three int values (requiring 12 bytes of memory) to a single int value representing the total number of seconds that have elapsed since midnight (requiring only 4 bytes of memory). If we make such a change, only the bodies of the methods that access the private data directly would need to change— in particular, the individual properties Hour, Minute and Second. There would be no need to modify the bodies of methods SetTime, ToUniversalString or ToString, because they do not access the private data directly. Designing the class in this manner reduces the likelihood of programming errors when altering the class’s implementation.

Similarly, each constructor could be written to include a copy of the appropriate statements from method SetTime. Doing so may be slightly more efficient, because the extra constructor call and the call to SetTime are eliminated. However, duplicating statements in multiple methods or constructors makes changing the class’s internal data representation more difficult and error-prone. Having one constructor call the other or even call SetTime directly requires any changes to SetTime’s implementation to be made only once.

Using Class Time2’s Overloaded Constructors

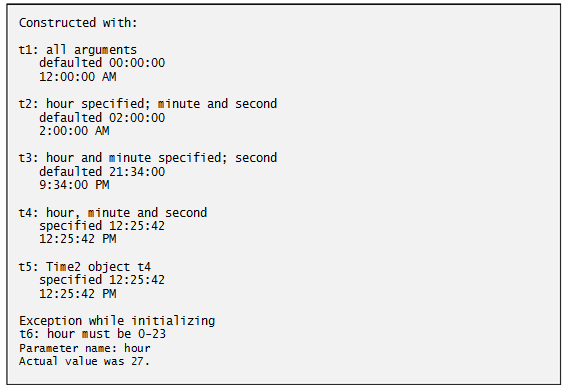




N.B. Remember to add to stop the console from closing at the end of the program.

Class Time2Test creates six Time2 objects (lines 13–18 and 44) to invoke the over-loaded Time2 constructors. Lines 13–18 demonstrate passing arguments to the Time2 constructors. C# invokes the appropriate overloaded constructor by matching the number and types of the arguments specified in the constructor call with the number and types of the parameters specified in each constructor declaration. Lines 13–16 each invoke the constructor at lines 16–19 of class Time2. Line 13 invokes the constructor with no arguments, which causes the compiler to supply the default value 0 for each of the three parameters. Line 14 invokes the constructor with one argument that represents the hour—the compiler supplies the default value 0 for the minute and second. Line 15 invokes the constructor with two arguments that represent the hour and minute—the compiler supplies the default value 0 for the second. Line 16 invokes the constructor with values for all three parameters. Line 17 invokes the constructor at lines 22–23 of class Time2. Lines 20–39 display the string representation of each initialized Time2 object to confirm that each was initialized properly.

Line 44 attempts to intialize t6 by creating a new Time2 object and passing three invalid values to the constructor. When the constructor attempts to use the invalid hour value to initialize the object’s Hour property, an ArgumentOutOfRangeException occurs. We catch this exception at line 46 and display its Message property.



Default and parameterless constructors

Every class must have at least one constructor. If you do not provide any constructors in a class’s declaration, the compiler creates a default constructor that takes no arguments when it’s invoked.

The compiler will **not** create a default constructor for a class that explicitly declares at least one constructor. In this case, if you want to be able to invoke the constructor with no arguments, **you must declare a parameterless constructor**—as in line 16 of class Time2. Like a default constructor, a parameterless constructor is invoked with empty parentheses. The Time2 parameterless constructor explicitly initializes a Time2 object by passing to SetTime 0 for each parameter. If we omit the parameterless constructor, clients of this class would not be able to create a Time2 object with the expression new Time2().