* Use Visual Studio Code to create a new .NET C# console application named **CSSapiens**.

Expand this hint for guidance on creating a new .NET C# console application.

* Open **Visual Studio Code** by selecting the desktop shortcut.
* In the Visual Studio Code Terminal window, enter the following command to create the new C# console application: dotnet new console -o CSSapiens.
* Press the **Enter** key to run the command.
* Run the newly created console application.

Expand this hint for guidance on running the new console application.

* In the Terminal window, run the following command to navigate to the **CSSapiens** application folder: cd D:\labfiles\CSSapiens.
* In the Terminal window, run the following command to run the new C# console application: dotnet run.
* Verify that the application runs successfully and outputs the following text in the console:

Expected requirement1 output



* Create a new file called Person.cs in the CSSapiens project to hold the new **Person** class.

Expand this hint for guidance on creating a new file.

* On the Visual Studio Code command bar, select **File**, and then select **New File** to create a new file.

You will normally create a new file in your application for each class that you want to use. This allows you to keep the code for each class in a separate space and makes it easier to read and understand.

* Save the new empty file as Person.cs in the D:\labfiles\CSSapiens folder.
* Begin defining the **Person** class by adding a using statement for the System namespace.

Expand this hint for guidance on referencing the System namespace.

* Add the following code at the top of the file:

using System;

The [System namespace](https://docs.microsoft.com/en-us/dotnet/api/system?view=net-5.0) contains fundamental **Microsoft.NET** classes that provide a range of in-built functionality for applications.

* Continue defining the **Person** class by adding a namespace statement for the CSSapiens namespace.In this Challenge Lab, you will define the Person class within the same C# [namespace](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/namespaces) as the one defined in your console application because, at this stage, the Person class will be working *only* with that application.

Expand this hint for guidance on referencing the CSSapiens namespace.

* Add the following code after the code already written:

namespace CSSapiens

{

}

* Continue defining the **Person** class by adding a public class statement and the name of the Person class.

Expand this hint for guidance on defining the Person class.

* Add the following code inside the braces of the CSSapiens namespace:

public class Person

{

}

[C# classes](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) act as containers for properties, methods, and events of objects. Other items may be contained in classes as well.

* Declare a private DateTime variable called birthDateAndTime that will be used in the Person class to hold each Person's date of birth.

Expand this hint for guidance on declaring the birthDateAndTime variable.

* Add the following code inside the braces of the Person class:

private DateTime birthDateAndTime;

The built-in [C# DateTime](https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=net-5.0) object provides ways that you can access the current date and time on the computer on which your application is running.

* Define a constructor for the **Person** class that causes your application to set the birthDateAndTime variable to today's date for each new instance of a **Person** object.

Expand this hint for guidance on how to define a constructor for the Person class.

* Add the following code below your birthDateAndTime variable declaration:

public Person()

{

this.birthDateAndTime = DateTime.Now;

}

When an instance of a class is created, the class [constructor](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-constructors) is executed. Constructors enable you to set default values and write code that is easy to read and flexible. A class can have more than one constructor, each taking different arguments. In this example, you will use the **default** constructor. No arguments are passed when it is executed.

The C# [**this**](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/this) keyword is used to refer to the **current instance** of the class.

* Save the **Person.cs** file.



* Declare a new private string variable called givenName.

Expand this hint for guidance on declaring the givenName variable.

* Add the following code below the birthDateAndTime variable declaration:

private string givenName;

* Declare a new private string variable called familyName.

Expand this hint for guidance on declaring the familyName variable.

* Add the following code below the givenName variable declaration:

private string familyName;

* Define a new readable/writable DateTime property called BirthDateAndTime that gets and sets the birthDateAndTime variable for the instance of the Person class.

Expand this hint for guidance on defining the BirthDateAndTime property.

* Add the following code below the familyName variable declaration:

public DateTime BirthDateAndTime

{

get {return this.birthDateAndTime;}

set {this.birthDateAndTime = value;}

}

[Properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/properties) provide a way to read, write, and calculate the value of a feature of an instance of an object created from a C# class definition. They can be read/write (as they are in your Person class definition), read-only, or write-only.

* Define a new readable/writable string property called GivenName that gets and sets the givenName variable for the instance of the Person class.

Expand this hint for guidance on defining the GivenName property.

* Add the following code below the BirthDateAndTime property definition:

public string GivenName

{

get {return this.givenName;}

set {this.givenName = value;}

}

* Define a new readable/writable string property called FamilyName that gets and sets the familyName variable for the instance of the Person class.

Expand this hint for guidance on defining the FamilyName property.

* Add the following code below the GivenName property definition:

public string FamilyName

{

get {return this.familyName;}

set {this.familyName = value;}

}

* Save the **Person.cs** file.



* Define a SetNames method that:

a) Takes two string parameters named givenName and familyName.  
b) Uses those parameters to set the values of the givenName and familyName variables for the instance of the Person class.  
c) Does not return a value.

Expand this hint for guidance on defining the SetNames method.

* Add the following code below the FamilyName property definition:

public void SetNames(string givenName,string familyName)

{

this.givenName = givenName;

this.familyName = familyName;

}

[Methods](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods) are blocks of code that contain C# statements. They are sometimes known by the names **sub-routine** or **function**. The statements are executed when the method is invoked by an application. Methods may or may not take arguments of different numbers and types, and may or may not return values to the application that invoked the method.  
  
In this Challenge Lab, you create a **SetNames** method for the Person class that takes two string arguments (the given name and the family name of a person), uses those arguments to set variables for the instance of the Person class (so that they become available through the class properties), and does *not* return a value.

* Save the **Person.cs** file.



* Declare a public event using the predefined C# EventHandler delegate called OnWake.

Expand this hint for guidance on declaring the OnWake event handler.

* Add the following code below the familyName variable declaration:

public event EventHandler OnWake;

* Declare a public event using the predefined C# EventHandler delegate called OnSleep.

Expand this hint for guidance on declaring the OnSleep event handler.

* Add the following code below the OnWake event declaration:

public event EventHandler OnSleep;

* Define a new method called Wake that raises the OnWake event.

Expand this hint for guidance on raising the OnWake event.

* Add the following code below the SetNames method definition:

public void Wake()

{

OnWake?.Invoke(this, EventArgs.Empty);

}

* Define a new method called Sleep that raises the OnSleep event.

Expand this hint for guidance on raising the OnSleep event.

* Add the following code below the Wake method definition:

public void Sleep()

{

OnSleep?.Invoke(this, EventArgs.Empty);

}

[Events](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events) provide a way for a class to tell other classes when something happens. A class that raises an event is called the **publisher**, and the classes that receive the event are called **subscribers**. In this challenge, you publish two events in the Person class named **OnWake** and **OnSleep**. You also create two methods in the Person class named **Wake** and **Sleep** that cause the new events to be raised. You will subscribe to the events in your console application in the next stage of the Challenge Lab.

The [C# EventArgs](https://docs.microsoft.com/en-us/dotnet/api/system.eventargs?view=net-5.0) class is one of the classes from the .NET System namespace and provides an object for you to use when you raise events. In this Challenge Lab, your code raises events that do not include any associated event data—which is why you used **EventArgs.Empty** in your code.

* Save the **Person.cs** file.



* In the Visual Studio Code Explorer, open the **Program.cs** file in your console application.
* Replace the code in the Main method with new code to create an instance of the Person class named person1.

Expand this hint for guidance on creating the instance of a Person object.

* Replace the code in the Main method of Program.cs with the following:

Person person1 = new Person();

Like many other languages, C# is an [object-oriented programming (OOP) language](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop). OOP has its focus on the objects that you want to use rather than the logic that is needed to use them. It is a programming approach that fits well with applications that are large, complicated, and actively updated (for example, Agile and DevOps approaches to software development).

You create instances of classes using the C# [new](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-initialize-objects-by-using-an-object-initializer) keyword rather than calling the class constructor directly.

* Use the SetNames method of the person1 instance passing a givenName parameter value of Fred and a familyName parameter value of Flintstone.

Expand this hint for guidance on using the SetNames method.

* Add the following code in the Main method of Program.cs

person1.SetNames("Fred", "Flintstone");

* Use the GivenName, FamilyName, and the BirthDateAndTime properties of the person1 instance to output a line in the console that looks like the example below:  
    
  The required output

Expand this hint for guidance on using properties.

* Add the following code in the Main method of **Program.cs**:

Console.WriteLine(person1.GivenName + " " + person1.FamilyName + "'s date of birth is " + person1.BirthDateAndTime.ToString("d"));

* After the use of the SetNames method, create an event handler called people\_OnWake to subscribe to the OnWake event of the person1 instance.Expand this hint for guidance on adding a pointer to the event handler for the OnWake event.
* Add the following code in the Main method of **Program.cs**:

person1.OnWake += people\_OnWake;

[Subscribing](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-subscribe-to-and-unsubscribe-from-events) to and unsubscribing from events is one of the key features in many programming approaches, including in C#.

* Create an event handler people\_OnSleep to subscribe to the OnSleep event of the person1 instance.

Expand this hint for guidance on adding a pointer to the event handler for the OnSleep event.

* Add the following code in the Main method of **Program.cs**:

person1.OnSleep += people\_OnSleep;

* Add an event handler in the Program class to handle the OnWake event for instances of the **Person** class.

This event handler should use the GivenName and FamilyName properties of the object that raised the event, and should output a line in the console that looks like the example below:

The expected output

Expand this hint for guidance on handling the OnWake event

* Add the following code to the Program class after the Main method:

static void people\_OnWake(object sender, EventArgs e)

{

Person p = sender as Person;

Console.WriteLine(p.GivenName + " " + p.FamilyName + " has woken up.");

}

The C# [as](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#as-operator) operator is used to convert the result of an expression to a given object type so that the features of the object type become available to your application.

* Add an event handler in the Program class to handle the OnSleep event for instances of the Person class.

This event handler should use the GivenName and FamilyName properties of the object that raised the event and should output a line in the console that looks like the example below:

The expected output

Expand this hint for guidance on handling the OnSleep event.

* Add the following code to the Program class after the Main method:

static void people\_OnSleep(object sender, EventArgs e)

{

Person p = sender as Person;

Console.WriteLine(p.GivenName + " " + p.FamilyName + " has fallen asleep.");

}

* Invoke the Wake method of the person1 instance.

Expand this hint for guidance on invoking the Wake method.

* Add the following code in the Main method of **Program.cs**:

person1.Wake();

* Invoke the Sleep method of the person1 instance.

Expand this hint for guidance on invoking the Sleep method.

* Add the following code in the Main method of **Program.cs**:

person1.Sleep();

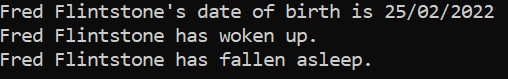
* Clean up your console application by freeing up resources for the person1 instance of the **Person** class now that you have finished using it.

Expand this hint for guidance on cleaning up the instance of the Person class.

* Add the following code in the Main method of **Program.cs**:

person1 = null;

* Save the **Program.cs** file.
* Run the .NET C# console application. Your application should now have output in the console that looks like the example below:





Summary

Congratulations, you have completed the **Understand the Fundamentals of C# Classes** Challenge Lab.

You have accomplished the following:

* You have created a new console application called CSSapiens.
* You have run the CSSapiens application.
* You have created a new file to hold the definition of the Person class.
* You have declared private variables to be used by instances of the Person class.
* You have defined a default constructor for the Person class that sets the birth date of an instance of the Person class to today.
* You have defined properties for instances of the Person class.
* You have defined methods for instances of the Person class.
* You have declared that instances of the Person class can raise events.
* You have written methods that raise events from an instance of the Person class.
* You have used properties and methods of an instance of the Person class in your console application.
* You have subscribed to events raised by an instance of the Person class in your console application.
* You have handled events raised by an instance of the Person class in your console application.
* You have used the C# null keyword to clean up an object instance.