Understanding encapsulation in Classes (C#)

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

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

* Open **Visual Studio Code** by double-clicking the desktop shortcut.
* In the Visual Studio Code Terminal window, type the following command which you will execute to create the new C# console application: dotnet new console -o CSPeople
* Press the **Enter** key to execute the command.
* Run the newly created console application.

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

* In the Terminal window, execute the following command to navigate to the **CSPeople** application folder: cd D:\labfiles\CSPeople
* In the Terminal window, execute 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:

The expected output

* Create a new file named Person.cs 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 make it easier to read and understand.

* Save the new empty file as Person.cs in the D:\labfiles\CSPeople folder.
* Begin defining the **Person** class by adding an 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 CSPeople namespace.

In this challenge, you will define the Person class within the same [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 CSPeople namespace.

* Add the following code after the code already written:

namespace CSPeople

{

}

* 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 CSPeople namespace:

public class Person

{

}

[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 can be contained in classes as well.

* Continue defining the **Person** class by declaring a private string variable named givenName.

Expand this hint for guidance on declaring the givenName variable.

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

private string givenName;

* Continue defining the **Person** class by declaring a private string variable named familyName.
* Define an overridable method named SayHello that does not return a value.

Expand this hint for guidance on defining the SayHello method.

* Add the following code below the familyName variable declaration:

public virtual void SayHello()

{

}

* Add code to the SayHello method that outputs in the console the phrase Hello from, followed by the givenName variable, a space, and then the familyName variable followed by the . character. The intended output should look like this **example**:

The expected output

Expand this hint for guidance on populating the SayHello method.

* Add the following code between the braces of the SayHello method:

Console.WriteLine("Hello from " + givenName + " " + familyName + ".");

* Save the **Person.cs** file.
* Create a new file to hold the new **Employee** class.
* Save the new empty file as Employee.cs in the D:\labfiles\CSPeople folder.
* Begin defining the **Employee** class by adding a using statement for the System namespace.
* Continue defining the **Employee** class by adding a namespace statement for the CSPeople namespace.
* Continue defining the **Employee** class by adding a public class statement, and the name of the Employee class. Ensure that the class inherits functionality from the Person class.

Expand this hint for guidance on defining the Employee class ensuring it inherits from the Person class.

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

public class Employee:Person

{

}

[Inheritance](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/inheritance) allows you to create new classes that reuse, add to, and/or change the way the other classes behave. The class whose functionalities are inherited is called the base class. The class that inherits those functionalities is called the derived class. You indicate that a derived class inherits functionality from a base class by adding a colon (**:**) and the name of the base class after the initial class definition. In C# a derived class can have only one direct base class.

* Continue defining the **Employee** class by declaring a private string variable named employer.
* Define a new readable/writable string property named Employer that gets and sets the employer variable for the instance of the Person class.

Expand this hint for guidance on defining the Employer property.

* Add the following code below the employer variable declaration:

public string Employer

{

get {return this.employer;}

set {this.employer = value;}

}

* Define a SayHello method that will override the definition in the **Person** class and output the phrase Hello from, followed by the givenName variable, a space, the familyName variable, the phrase works at, the employer variable, and the (.) character. The intended output should look like the following example:

The expected output

Expand this hint for guidance on defining the SayHello method that will override the definition in the Person class.

* Add the following code to your class definition:

public override void SayHello()

{

Console.WriteLine("Hello from " + this.givenName + " " + this.familyName + " who works at " + this.employer + ".");

}

* Save the **Employee.cs** file.
* Open the **Program.cs** file from the D:\labfiles\CSPeople folder.
* Delete from the Main method in Program.cs the code that writes output in the console.
* Write code to create an instance of the Person class named person1.

Expand this hint for guidance on creating an instance of the Person class.

* Add the following code to the Main method:

Person person1 = new Person();

* Write code to create an instance of the Employee class named employee1.

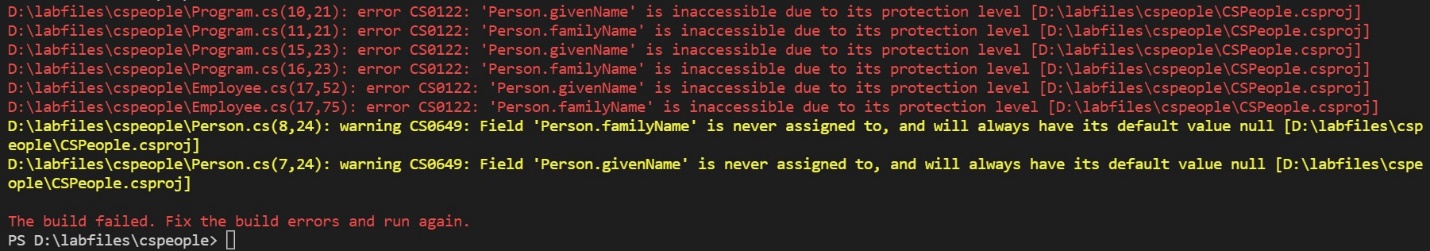
Expand this hint for guidance on creating an instance of the Employee class.

* Add the following code to the Main method:

Employee employee1 = new Employee();

When an instance of a class is created, the class **constructor** is executed. [C# constructors](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-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 haven't written a constructor, so the 'default' constructors of the Person class and the Employee class are executed to create the **person1** and **employee1** instances.

* Save the Program.cs file.
* Run your application.The expected output should show that the build has failed for the application and that the givenName and familyName variables are inaccessible due to their protection levels as shown below:



* Open the **Person.cs** file.
* Change the variable declarations in the Person class so that the givenName and familyName are declared as public variables.

Expand this hint for guidance on declaring public variables.

* **Replace** the givenName and familyName variable declarations in the Person class definition with the following code:
* public string givenName;

public string familyName;

As a programmer, you will give all C# classes and their functionality an **accessibility level** by using [access modifiers](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifier/). The accessibility level controls whether the class and/or its functionality can be used from code in your application or other applications.  
  
When you use the **public** keyword, your code can be accessed from anywhere. When you use the **private** keyword the code is only accessible within the class. You will examine other access modifiers later in this challenge.  
  
Using **access modifiers** provides a way for you to hide functionality that isn't intended to be used from outside your class or application. This concept is called [encapsulation](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/) and it helps you to reduce the possibility of coding errors or malicious exploits.

* Save the Person.cs file.
* Return to the **Program.cs** file, and then write code that sets the givenName variable of the person1 instance to the value Wilma.

Expand this hint for guidance on setting the givenName variable.

* Add the following code to the Main method:

person1.givenName = "Wilma";

* Write code that sets the familyName variable of the person1 instance to the value Flintsone.
* Write code to invoke the SayHello method of the person1 instance.

Expand this hint for guidance on invoking the SayHello method.

* Add the following code to the Main method:

person1.SayHello();

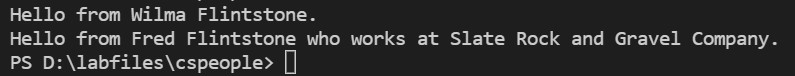
* After the code that creates the employee1 instance, write code that sets the givenName variable of the employee1 instance to the value Fred.
* Write code that sets the familyName variable of the employee1 instance to the value Flintstone.
* Write code that sets the Employer property of the employee1 instance to the value Slate Rock and Gravel Company.

Expand this hint for guidance on setting the Employer property.

* Add the following code to the Main method:

employee1.Employer = "Slate Rock and Gravel Company";

* Write code to invoke the SayHello method of the employee1 instance.
* Save the Program.cs file.
* Run your application. The expected output is shown below:



* Open the Person.cs file.
* Change the variable declarations in the Person class so that the givenName and familyName are declared as internal variables.

Expand this hint for guidance on declaring internal variables.

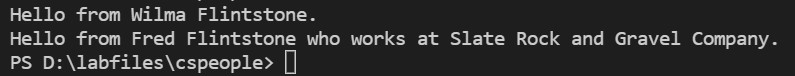
* **Replace** the givenName and familyName variable declarations in the Person class definition with the following code:

internal string givenName;

internal string familyName;

As a programmer, you will use the internal access modifier so that functionality is only allowed to code within the same assembly. An assembly is a collection of C# classes and resources that are built to work together and form a logical unit of functionality.

* Save the Person.cs file.
* Run your application. The expected output is shown below:



* Change the variable declarations in the Person class so that the givenName and familyName are declared as protected variables.

Expand this hint for guidance on declaring protected variables.

* **Replace** the givenName and familyName variable declarations in the Person class definition with the following code:  
    
  protected string givenName; protected string familyName;

As a programmer, you will use the protected access modifier so that functionality is only allowed to code within the same class and its derived classes.

* Save the Person.cs file.
* Run your application. The expected output is shown below:

A screen shot of a computer program

Description automatically generated