Working with Polymorphism

* Create a new file called StableIsotope.cs to hold the new **StableIsotope** 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 the new file.

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

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

In this challenge, you will define the StableIsotope 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 StableIsotope class will be working ONLY with that application.

Expand this hint for guidance on referencing the CSElements namespace.

* Add the following code after the code already written:  
    
  namespace CSElements
* Continue defining the **StableIsotope** class by adding a public class statement and the name of the StableIsotope class.

Expand this hint for guidance on defining the StableIsotope class.

* Add the following code inside the braces of the CSElements namespace:  
    
  public class StableIsotope

[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.

* Continue defining the **StableIsotope** class by declaring a private string variable named name.

Expand this hint for guidance on declaring the name variable.

* Add the following code inside the braces of the StableIsotope class:  
    
  private string name;
* Continue defining the **StableIsotope** class by declaring a private integer variable named protons.

Expand this hint for guidance on declaring the protons variable.

* Add the following code beneath the name variable declaration:  
    
  private int protons;
* Continue defining the StableIsotope class by declaring a private integer variable named neutrons.
* Continue defining the StableIsotope class by declaring a private integer variable named electrons.
* Define a new readable/writable string property named Name that gets and sets the name variable for the instance of the StableIsotope class.

Expand this hint for guidance on defining the Name property.

* Add the following code below the electron’s variable declaration:  
    
  public string Name { get {return this.name;} set {this.name = 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, read-only, or write-only.

* Define a new readable/writable integer property named Protons that gets and sets the protons variable for the instance of the StableIsotope class.
* Define a new readable/writable integer property named Neutrons that gets and sets the neutrons variable for the instance of the StableIsotope class.
* Define a new readable/writable integer property named Electrons that gets and sets the electrons variable for the instance of the StableIsotope class.
* Define a method called Report that returns a string that shows the name of the instance of the object followed by the number or protons, neutrons, electrons, and the values in the following table. For any property that is not specified, use the default value.

| **Property** | **Value** |
| --- | --- |
| Method Name | Report |
| name | is an isotope that has protons x |
| protons | , neutrons x |
| neutrons | + and electrons x+ |
| electrons | "." |

Expand this hint for guidance on defining the Report method.

* Add the following code below the Electrons property definition:  
    
  public string Report() { return this.name + " is an isotope that has protons x" + this.protons + ", neutrons x" + this.neutrons + " and electrons x" + this.electrons + "."; }

[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 you create a **Report** method for the StableIsotope class that uses the variables declared in the StableIsotope class and returns a string for display in the console.

* Save the **StableIsotope.cs** file.
* Open the **Program.cs** file, and then view the Main method.
* Delete the line of code that outputs **Hello World!** in the console.
* Write code to declare an integer variable named atomicMass.

Expand this hint for guidance on declaring the atomicMass variable.

* Add the following code in **Program.cs**:  
    
  int atomicMass;
* Write code to create an instance of the StableIsotope class named sodium.

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

* Add the following code below the atomicMass variable declaration:  
    
  StableIsotope sodium = new StableIsotope();
* Write code to set the value of the Protons property for the sodium instance to the value 11.

Expand this hint for guidance on setting the Protons property.

* Add the following code below the creation of the sodium instance:  
    
  sodium.Protons = 11;
* Set the value of the Neutrons property for the sodium instance to the value 12.
* Set the value of the Electrons property for the sodium instance to the value of the Protons property of the instance.

Expand this hint for guidance on setting the Electrons property.

* Add the following code below setting the Neutrons property:  
    
  sodium.Electrons = sodium.Protons;
* Set the value of the atomicMass variable to the Protons property value of the sodium instance plus the Neutrons property value of the sodium instance.

Expand this hint for guidance on setting the value of the atomicMass variable.

* Add the following code below setting the Neutrons property:  
    
  atomicMass = sodium.Protons + sodium.Neutrons;
* Set the value of the Name property value of the sodium instance to the value Sodium plus the atomicMass variable.

Expand this hint for guidance on setting the value of the Name property.

* Add the following code below setting the atomicMass variable:  
    
  sodium.Name = "Sodium" + atomicMass;
* Output the result of invoking the Report method of the sodium instance to the console.

Expand this hint for guidance on how to output the result of invoking the Report method.

* Add the following code below setting the Name property:  
    
  Console.WriteLine(sodium.Report());
* Save the **Program.cs** file.
* Run the console application. The expected output is shown below:  
    
  The expected output
* Create a new file to hold the new **RadioactiveIsotope** class.
* Save the new empty file as RadioactiveIsotope.cs in the D:\labfiles\CSElements folder.
* Begin defining the **RadioactiveIsotope** class by adding a using statement for the System namespace.
* Continue defining the **RadioactiveIsotope** class by adding a namespace statement for the CSElements namespace.
* Continue defining the **RadioactiveIsotope** class by adding a public class statement and the name of the RadioactiveIsotope class. *Ensure that the class inherits functionality from the StableIsotope class*.

Expand this hint for guidance on defining the RadioactiveIsotope class ensuring it inherits from the StableIsotope class.

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

public class RadioactiveIsotope:StableIsotope

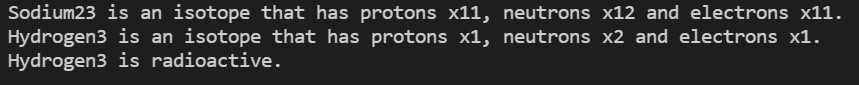
{

}

[Inheritance](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/inheritance) lets you 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#, unlike some other programming languages, a derived class can have only one direct base class.

* Define a method named Decay in the RadioactiveIsotope class that returns the string Name property value of the object instance of the class followed by the phrase is radioactive., followed by a new line character. If you're wondering where the Name property is defined, it comes from the StableIsotope class, which your new class inherits.

Expand this hint for guidance on defining the Decay method.

* Add the following code in the class definition:  
    
  public string Decay() { return this.Name + " is radioactive." + "\n"; }
* Save the RadioactiveIsotope.cs file.
* Return to the Main method of your console application in the **Program.cs** file.
* Create an instance of the RadioactiveIsotope class called tritium.
* Set the Protons property of the tritium instance to the value 1.
* Set the Neutrons property of the tritium instance to the value 2.
* Set the Electrons property of the tritium instance to the value 1.
* Set the value of the atomicMass variable to the value of the Protons property of the tritium instance **PLUS** the value of the Neutrons property of the tritium instance.
* Set the tritium.Name to “Hydrogen ”+ atomicMass.
* Output in the console the result of invoking the Report method of the tritium instance.
* Output in the console the result of invoking the Decay method of the tritium instance.
* Save the **Program.cs** file.
* Run your console application. You should see output in the console as shown below:  
    
  

---------------------------------------------------------

* Create a new file to hold the new AlphaDecayIsotope class.
* Save the new empty file as AlphaDecayIsotope.cs in the D:\labfiles\CSElements folder.
* Create a new file to hold the new BetaDecayIsotope class.
* Save the new empty file as BetaDecayIsotope.cs in the D:\labfiles\CSElements folder.
* Create a new file to hold the new GammaDecayIsotope class.
* Save the new empty file as GammaDecayIsotope.cs in the D:\labfiles\CSElements folder.
* In the AlphaDecayIsotope class file, create an AlphaDecayIsotope class definition, adding necessary using, namespace, and public class statements. Ensure that the class inherits functionality from the RadioactiveIsotope class.

Expand this hint for guidance on creating the AlphaDecayIsotope class definition.

* Add the following code to the AlphaDecayInstance.cs file:

using System;

namespace CSElements

{

public class AlphaDecayIsotope:RadioactiveIsotope

{

}

}

* In the BetaDecayIsotope class file, create a BetaDecayIsotope class definition, adding necessary using, namespace, and public class statements. Ensure that the class inherits functionality from the RadioactiveIsotope class.
* In the GammaDecayIsotope class file, create a GammaDecayIsotope class definition, adding necessary using, namespace, and public class statements. Ensure that the class inherits functionality from the **RadioactiveIsotope** class.
* Save each of your new files.
* In the class definition for the RadioactiveIsotope class, change the Decay method so that it can be used in polymorphic ways.

Expand this hint for guidance on changing the Decay method so that it can be used in polymorphic ways.

* Add the following code to change the Decay method signature so that it includes the virtual keyword:  
    
  public virtual string Decay()

In [polymorphism](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/polymorphism) , base classes define virtual functionality and derived classes can override that functionality, which means they provide their own definitions.

* Save the **RadioactiveIsotope.cs** file.
* In the **AlphaDecayIsotope** class definition, define a Decay method that will override the definition in the **RadioactiveIsotope** class and output the Name of the object instance followed by the phrase “is radioactive and undergoes alpha decay.”, followed by a new line.

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

* Add the following code to your class definition:  
    
  public override string Decay() { return this.Name + " is radioactive and undergoes alpha decay." + "\n"; }
* In the BetaDecayIsotope class definition, define a Decay method that will override the definition in the RadioactiveIsotope class, and then output the Name of the object instance followed by the phrase is radioactive and undergoes beta decay. followed by a new line.
* In the GammaDecayIsotope class definition, define a Decay method that will override the definition in the RadioactiveIsotope class, and then output the Name of the object instance followed by the phrase is radioactive and undergoes gamma decay. followed by a new line.
* Return to the **Program.cs** file in your console application.
* Create an instance of the AlphaDecayIsotope class named uranium, and then assign values to the properties of the uranium as shown in the table below:

| **Property** | **Value** |
| --- | --- |
| Protons | 92 |
| Neutrons | 146 |
| Electrons | 92 |
| Name | Uranium238 |

Expand this hint for guidance on creating the uranium instance and setting properties.

* Add the following code to **Program.cs**:  
    
  AlphaDecayIsotope uranium = new AlphaDecayIsotope();

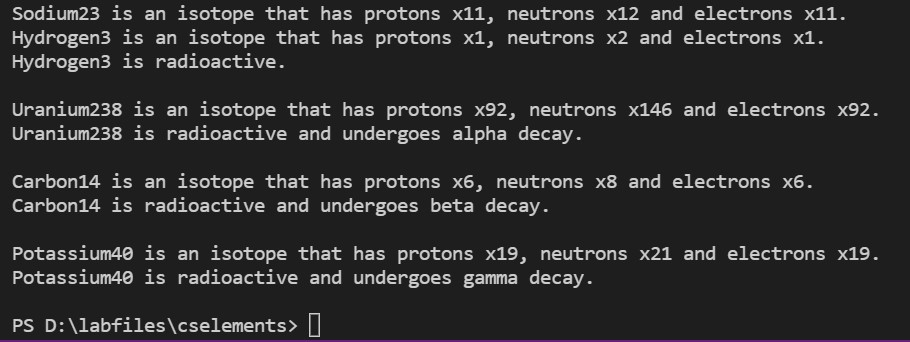
uranium.Protons = 92; uranium.Neutrons = 146; uranium.Electrons = 92; uranium.Name = "Uranium238";

* Create an instance of the BetaDecayIsotope class named carbon, and then assign values to the properties of the carbon instance as shown in the table below:

| **Property** | **Value** |
| --- | --- |
| Protons | 6 |
| Neutrons | 8 |
| Electrons | 6 |
| Name | Carbon14 |

* Create an instance of the GammaDecayIsotope class named potassium, and then assign values to the properties of the potassium instance as shown in the table below:

| **Property** | **Value** |
| --- | --- |
| Protons | 19 |
| Neutrons | 21 |
| Electrons | 19 |
| Name | Potassium40 |

* Add a line of code to output in the console the result of invoking the Report method of the uranium object instance.
* Add a line of code to output in the console the result of invoking the Decay method of the uranium object instance.
* Add a line of code to output in the console the result of invoking the Report method of the carbon object instance.
* Add a line of code to output in the console the result of invoking the Decay method of the carbon object instance.
* Add a line of code to output in the console the result of invoking the Report method of the potassium object instance.
* Add a line of code to output in the console the result of invoking the Decay method of the potassium object instance.
* Save all your files.
* Run your console application. You should see output in the console as shown below:  
    
  

Congratulations, you have completed the **Understand C# Class Inheritance and Polymorphism** Challenge Lab.

You have accomplished the following:

* Created a new console application.
* Created a class representation of a StableIsotope object.
* Created a class representation of a RadioactiveIsotope object that inherits from the StableIsotope class.
* Created class representations of an AlphaDecayIsotope object, a BetaDecayIsotope object, and a GammaDecayIsotope object which all inherit from the RadioactiveIsotope class.
* Used polymorphism to modify the way derived classes behave.