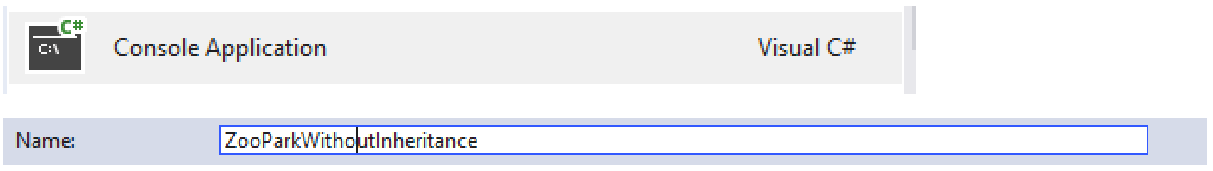
EXERCISE 1

The tasks in the first part of this section provide step by step instructions on how to use inheritance to create specific classes that is built upon a base class. This is followed by a number of tasks for you to complete by applying the knowledge that you have previously gained.

TASK 1: Zoo Park (Without Inheritance)

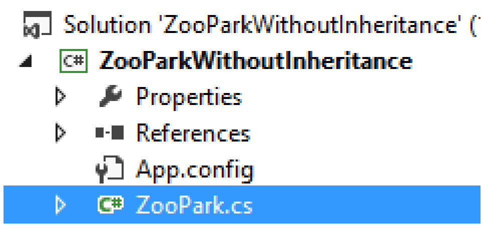
This task requires you to create a console application which will print the details of several animals in a zoo. We will write a Console Application to achieve this.

Step 1: Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Console Application and name it `e.g. ZooParkWithoutInheritance`

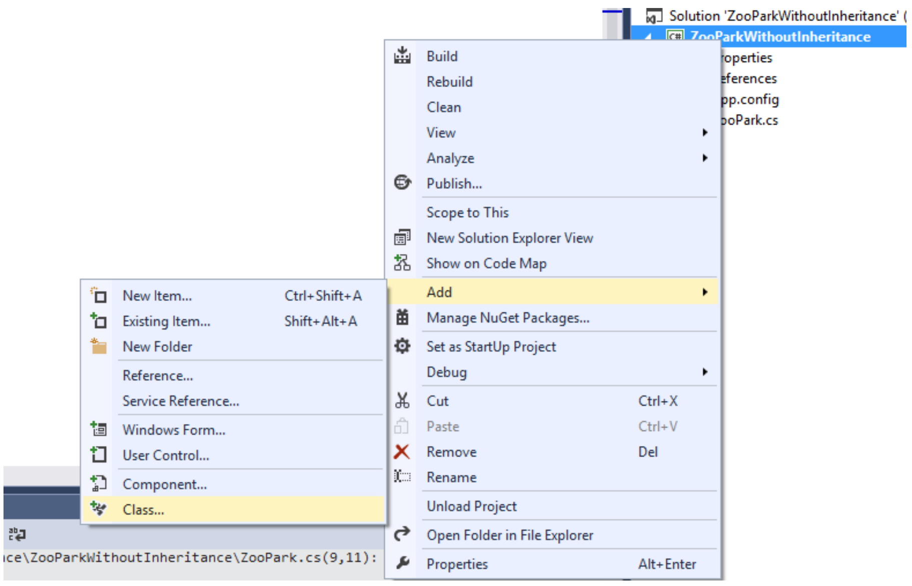

Console Application Visual C#

Name: ZooParkWithoutInheritance

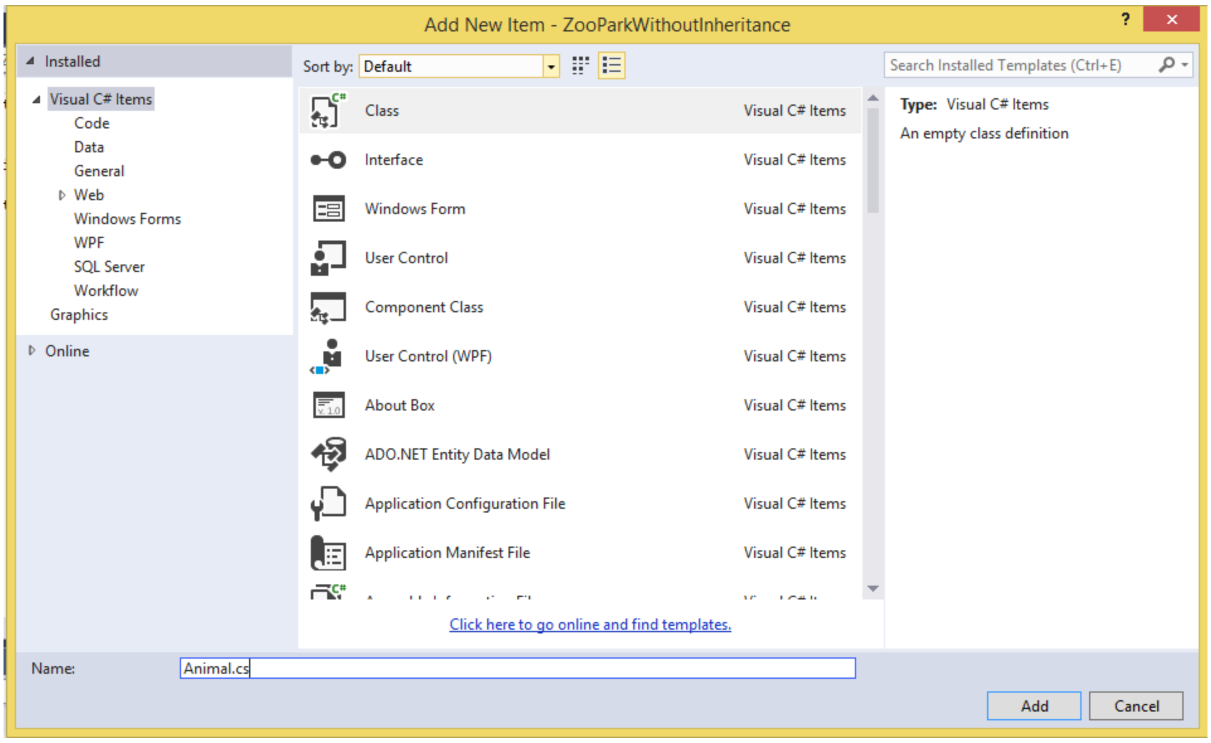
Step 2: Right Click on Program.cs in the Solution explorer and rename to ZooPark.cs and select Yes in the followed popup:


Solution 'ZooParkWithoutInheritance' (
ZooParkWithoutInheritance
Properties
References
App.config
ZooPark.cs

Step 3: As well as the ZooPark class which is where you will create the animal objects and print out the details of them to the console, we need to create an animal class. Right click on the project, select `Add` and click `Class` to add a new class:



Name your new class `Animal.cs` and select `Add`:


Add New Item - ZooParkWithoutInheritance

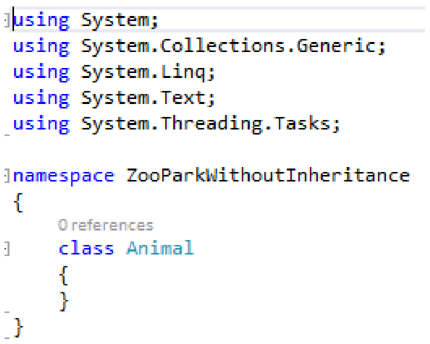
Class	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
Component Class	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File	Visual C# Items

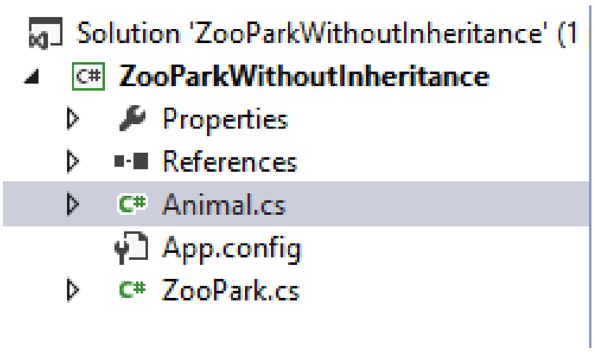
Type: Visual C# Items
An empty class definition

Click here to go online and find templates.

Name: Animal.cs

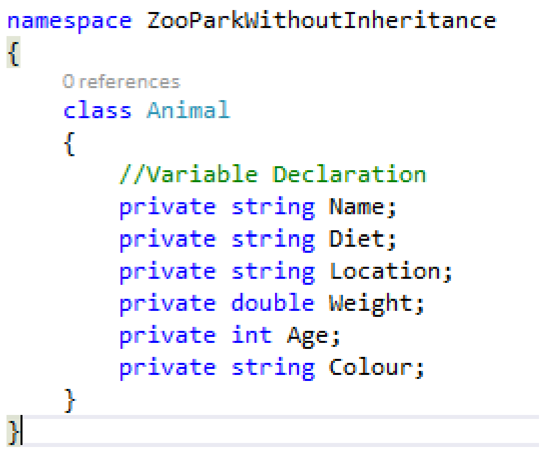
This will create blank Animal class and add it to your project:


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ZooParkWithoutInheritance
{
    0 references
    class Animal
    {
    }
}
```


Solution 'ZooParkWithoutInheritance' (1
ZooParkWithoutInheritance
Properties
References
Animal.cs
App.config
ZooPark.cs

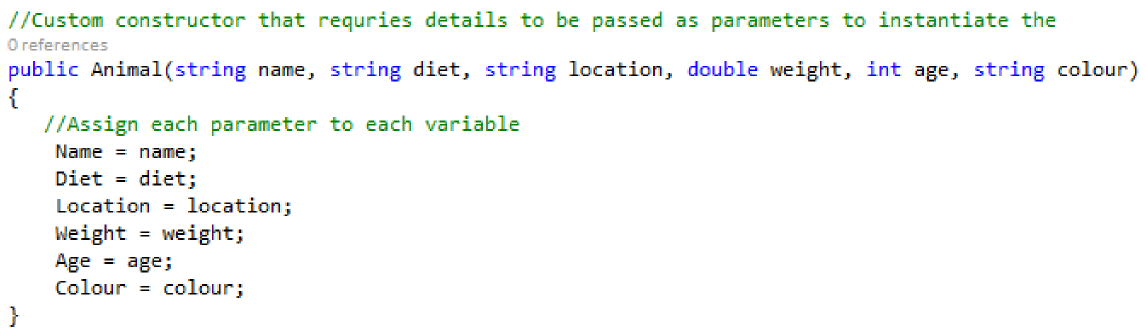
This class will be a blue print for what the animals in your zoo (objects) will contain. We will now add several variables and methods to the Animal class.

First and foremost, we will declare the attributes you would like the animal to have e.g. name, age etc:



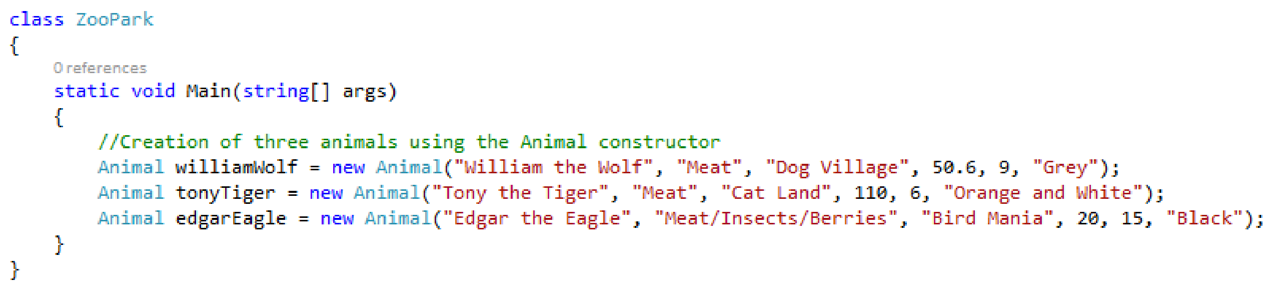
```
namespace ZooParkWithoutInheritance
{
    0 references
    class Animal
    {
        //Variable Declaration
        private string Name;
        private string Diet;
        private string Location;
        private double Weight;
        private int Age;
        private string Colour;
    }
}
```

Now we will add a constructor to the Animal class that will allow you to create an instance of the Animal class and allow the details of the Animal to be passed as variables to the constructor that will assign the class variables to the parameters values:



```
//Custom constructor that requries details to be passed as parameters to instantiate the
0 references
public Animal(string name, string diet, string location, double weight, int age, string colour)
{
   //Assign each parameter to each variable
    Name = name;
    Diet = diet;
    Location = location;
    Weight = weight;
    Age = age;
    Colour = colour;
}
```

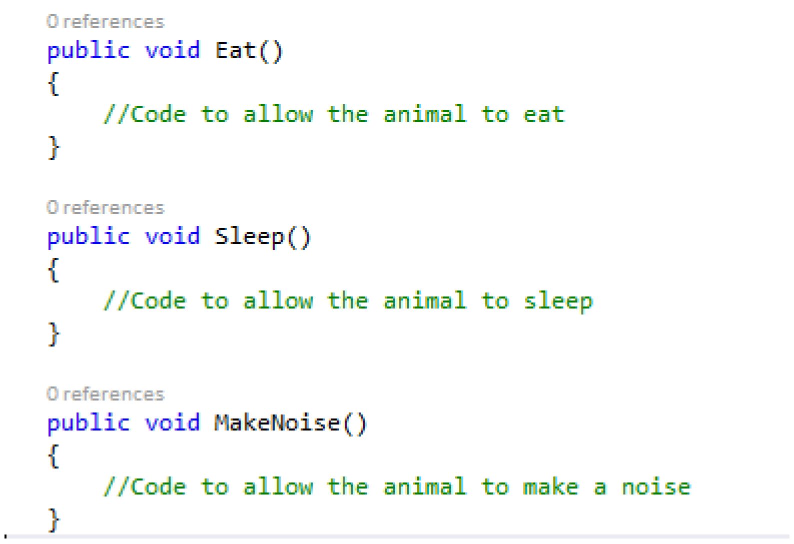
We are now at the stage that we can create Animal objects in our ZooPark class so we will create several animals:



```
class ZooPark
{
    0 references
    static void Main(string[] args)
    {
        //Creation of three animals using the Animal constructor
        Animal williamWolf = new Animal("William the Wolf", "Meat", "Dog Village", 50.6, 9, "Grey");
        Animal tonyTiger = new Animal("Tony the Tiger", "Meat", "Cat Land", 110, 6, "Orange and White");
        Animal edgarEagle = new Animal("Edgar the Eagle", "Meat/Insects/Berries", "Bird Mania", 20, 15, "Black");
    }
}
```

We will know add methods to the Animal class that will allow each animal to perform several coomon actions:

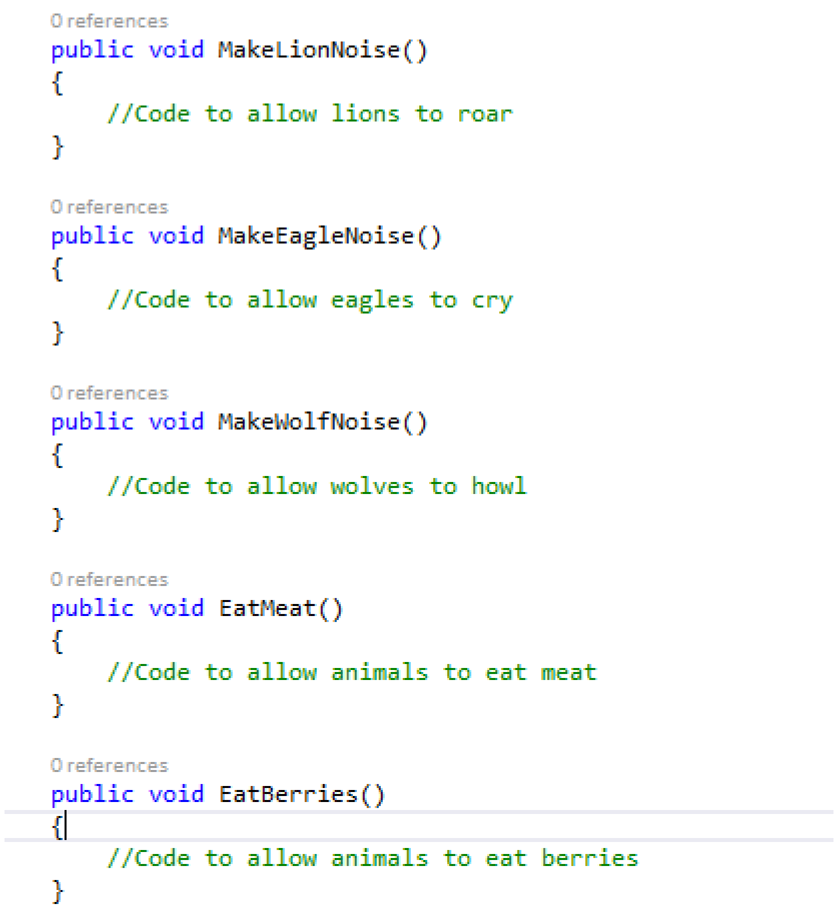
- `Sleep`(): a method to make the animal lie down and take a nap
- `Eat`(): a method to make the animal eat
- `MakeNoise`(): a method to allow the animal to make a sound



```
0 references
public void Eat()
{
    //Code to allow the animal to eat
}

0 references
public void Sleep()
{
    //Code to allow the animal to sleep
}

0 references
public void MakeNoise()
{
    //Code to allow the animal to make a noise
}
```

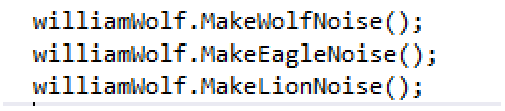
Our class now allows the animals to eat, sleep and make a noise. However, what if the animal requires to perform an action that is specify to the animal e.g. each animal makes a different noise and eats different food.

We will now add these methods into our class:



```
0 references
public void MakeLionNoise()
{
    //Code to allow lions to roar
}

0 references
public void MakeEagleNoise()
{
    //Code to allow eagles to cry
}

0 references
public void MakeWolfNoise()
{
    //Code to allow wolves to howl
}

0 references
public void EatMeat()
{
    //Code to allow animals to eat meat
}

0 references
public void EatBerries()
{
    //Code to allow animals to eat berries
}
```

As you can see now the class is becoming quite cluttered and contains methods that not animals need or will use. For example, wolves will not make an eagle noise or a lion noise but the animal class contains methods to allow this:



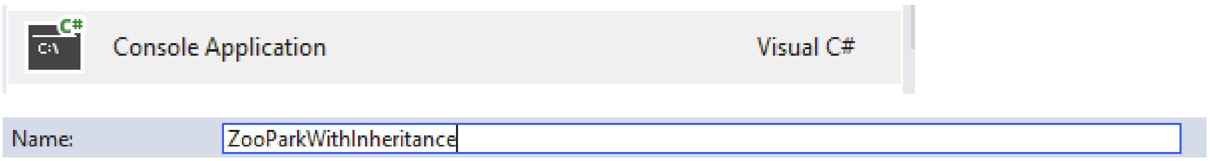
```
williamWolf.MakeWolfNoise();
williamWolf.MakeEagleNoise();
williamWolf.MakeLionNoise();
```

We will now use inheritance to create a base Animal class which will give the animals everything they have in common e.g. Name, Make a Noise and then create sub classes that inherit from the base class that will contain more specific behaviours. E.g. Tiger class that inherits from the Animal class.

TASK 2: Zoo Park with Inheritance

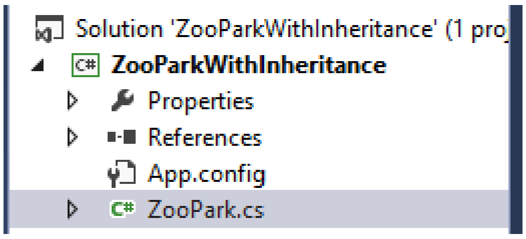
This task requires you to create a console application which will print the details of several animals in a zoo using inheritance. We will write a Console Application to achieve this.

Step 1: Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Console Application and name it `e.g. ZooParkWithInheritance`

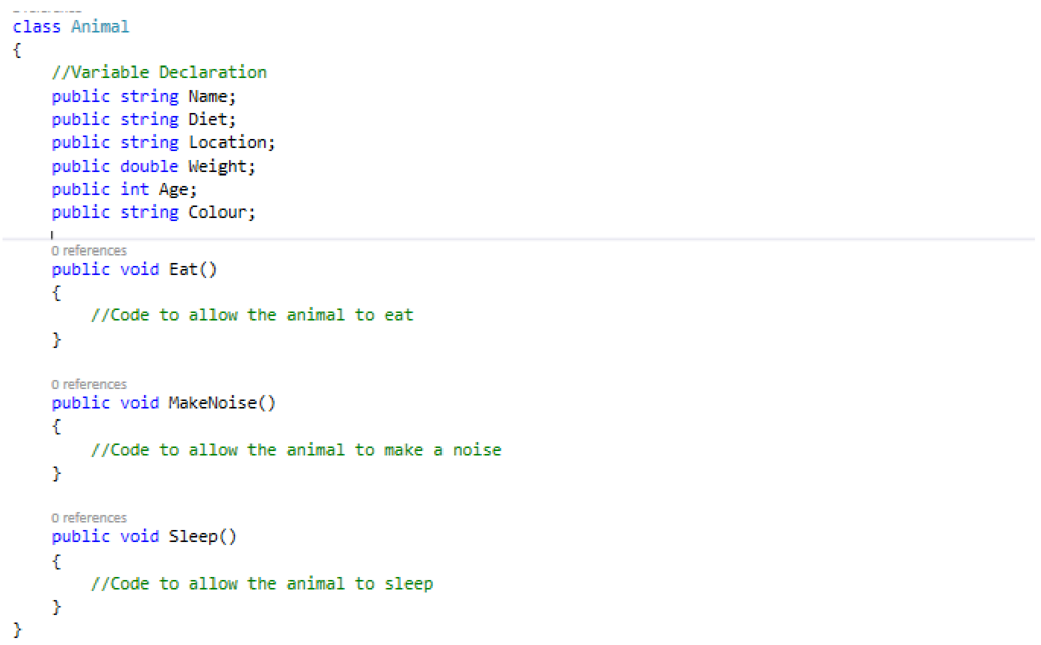

Console Application Visual C#

Name: ZooParkWithInheritance

Step 2: Right Click on Program.cs in the Solution explorer and rename to ZooPark.cs and select Yes in the followed popup:



Step 3: Create a base `Animal` class similar to the Animal class in Task 1:


```
class Animal
{
    //Variable Declaration
    public string Name;
    public string Diet;
    public string Location;
    public double Weight;
    public int Age;
    public string Colour;

    0 references
    public void Eat()
    {
        //Code to allow the animal to eat
    }

    0 references
    public void MakeNoise()
    {
        //Code to allow the animal to make a noise
    }

    0 references
    public void Sleep()
    {
        //Code to allow the animal to sleep
    }
}
```

This base class contains all of the common attributes and behaviours that each animal wil have.

The Zoo contains several species of animal including Tigers, Lions, Eagles, Wolves and Hippos.

We will now have to create a class for each specific animal that will inherit from the Animal base class.

Step 4: Create a Lion class that inherits from the base Animal class. To do this, add a new class called `Lion.cs` which will create a blank class:

```
namespace ZooParkWithInheritance
{
    0 references
    class Lion
    {
    }
}
```

To allow a class to inherit from a base class, the syntax is as follows:

```
class SubClass : BaseClass
{
}
```

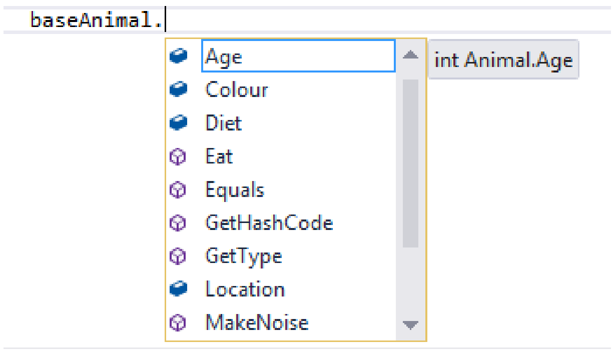
Therefore to allow the Lion to inherit from the Animal class, change the class definition to the following:

```
//Lion now inherits all public attributes and methods from the Animal base class
0 references
class Lion : Animal
{

}
```

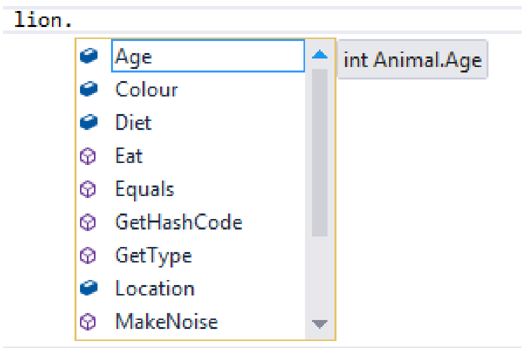
Let's try to create an instance of each class to see if the Lion class does in fact inherit from the Animal class. In ZooPark.cs, create an Animal and a Lion object:

```
static void Main(string[] args)
{
    //Creating base Animal object
    Animal baseAnimal = new Animal();

    //Creating Lion object that inherits from Animal
    Lion lion = new Lion();
}
```

If you try to access the public methods and variables of each, you can see that Lion has inherited all of the public methods and attributes as shown below:

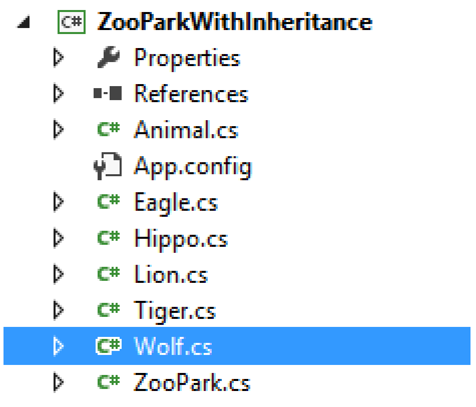


```
baseAnimal.
```



```
lion.
```

Step 5: Create classes for a `Tiger`, a `Wolf`, an `Eagle` and a `Hippo` that all inherit from the base Animal class.



```
ZooParkWithInheritance
    Properties
    References
    Animal.cs
    App.config
    Eagle.cs
    Hippo.cs
    Lion.cs
    Tiger.cs
    Wolf.cs
    ZooPark.cs
```

```
//Creating base Animal object
Animal baseAnimal = new Animal();

Lion lion = new Lion();
Eagle eagle = new Eagle();
Hippo hippo = new Hippo();
Tiger tiger = new Tiger();
Wolf wolf = new Wolf();
```

Now you have subclasses for each animal that all inherit from the base Animal class.

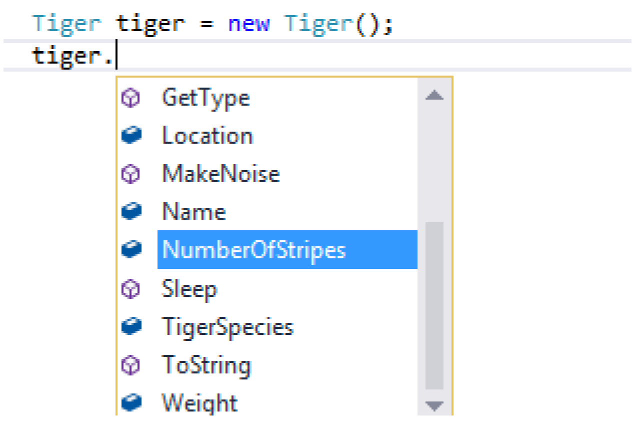
The idea of inheritance implements the **IS-A** relationship. For example, Lion **IS-A** animal and Wolf **IS-A** Animal.

Step 6: We now have the classes for each animal so it is time to add some attributes which are unique to that animal. We will use the Tiger as an example.

Open the `Tiger` class and add attributes that only a tiger will have out of your animals, for example, number of stripes or the species of Tiger:

```
//Tiger now inherits all public attributes and methods from the Animal base class
2 references
class Tiger : Animal
{
    //Declaration of Attributes unique to a Tiger
    public string TigerSpecies;
    public int NumberOfStripes;
}
```

Now if you try to access the public methods and variables of a Tiger object, you can see that Tiger has inherited all of the public methods and attributes of Animal but also has its own specific attributes as shown below:



```
Tiger tiger = new Tiger();
tiger.
```

```
GetType
Location
MakeNoise
Name
NumberOfStripes
Sleep
TigerSpecies
ToString
Weight
```

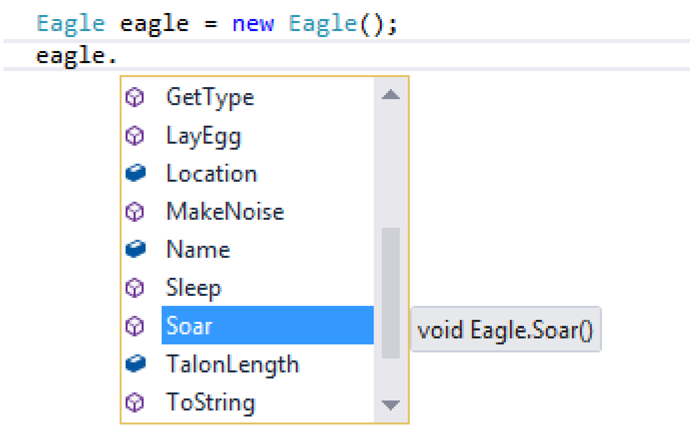
Step 7: Now add specific attributes to each of the subclasses of Animal e.g. Wolf can have an average howl length etc.

Step 8: We now have the classes for each animal with specific Attributes so it is time to add some methods which are unique to that animal. We will use the Eagle as an example.

Open the `Eagle` class and add methods that only an Eagle will have out of your animals, for example, LayEgg:

```
//Eagle now inherits all public attributes and methods from the Animal base class
2 references
class Eagle : Animal
{
    //Declaration of Attributes unique to a Eagle
    public string EagleSpecies;
    public double WingSpan;
    public double TalonLength;

    0 references
    public void LayEgg()
    {
        //Code to allow Eagle to Lay an Egg
    }

    0 references
    public void Fly()
    {
        //Code to allow Eagle to Fly
    }

    0 references
    public void Soar()
    {
        //Code to allow Eagle to Soar
    }
}
```

Now if you try to access the public methods and variables of an Eagle object, you can see that Eagle has inherited all of the public methods and attributes of Animal but also has its own specific methods and attributes as shown below:



```
Eagle eagle = new Eagle();
eagle.
```

Step 9: Now add specific Methods to each of the subclasses of Animal e.g. join pack to hunt

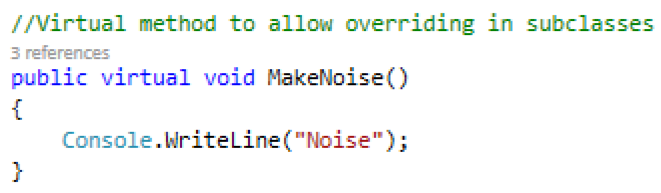
TASK 3: Different Animals make different noises

Tigers roar, wolves howl and as far we know hippo's don't make any sound at all. Each of the classes that inherit from Animal will a have `MakeNoise`() method, but each of those methods will work a different way and have different code.

When a subclass changes the behaviour of one of the methods that it inherited, we say that it **`overrides`** the method.

So when you have a subclass that inherits from a base class, it must inherit all of the base behaviours but you can modify them in the subclass so they are not performed exactly the same way. That's what overriding is all about.

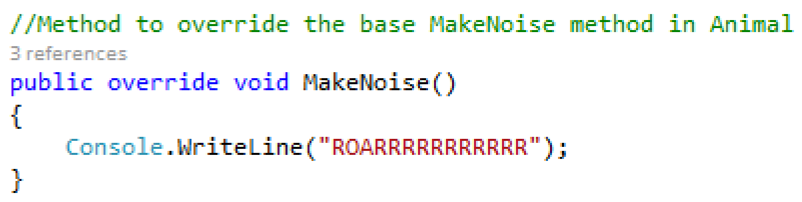
Step 1: Continue working on your project from Task 2 from Exercise 1 and we will use the Tiger Class as an example. To allow the base methods to be overridden, the base methods must be marked as Virtual. A **`virtual`** method can be redefined. The virtual keyword designates a method that is overridden in derived classes. Open the Animal class and mark the `MakeNoise`() method as Virtual as shown below:



```
//Virtual method to allow overriding in subclasses
3 references
public virtual void MakeNoise()
{
    Console.WriteLine("Noise");
}
```

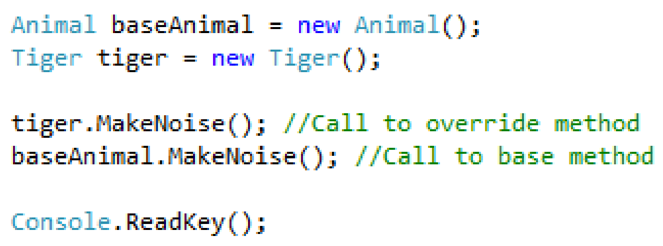
Step 2: Since the base `MakeNoise`() method is now virtual, we can override this method in the Tiger class to allow the Tiger Object to ROAARRRR when the `MakeNoise`() method is called.

To do this, create a `MakeNoise`() method in the Tiger class and include the **`override`** keyword in the method declaration as shown below:

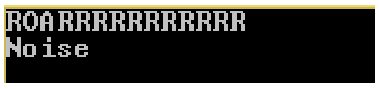


```
//Method to override the base MakeNoise method in Animal
3 references
public override void MakeNoise()
{
    Console.WriteLine("ROARRRRRRRRRRR");
}
```

Step 3: Call the `MakeNoise`() method on a Tiger object and an Animal object as shown below:



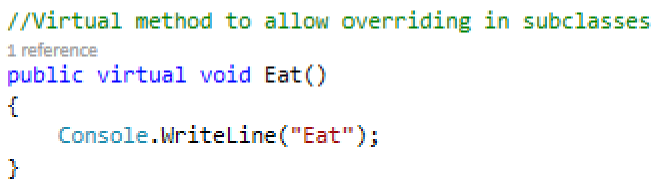
```
Animal baseAnimal = new Animal();
Tiger tiger = new Tiger();

tiger.MakeNoise(); //Call to override method
baseAnimal.MakeNoise(); //Call to base method

Console.ReadKey();
```



```
ROARRRRRRRRRRR
Noise
```

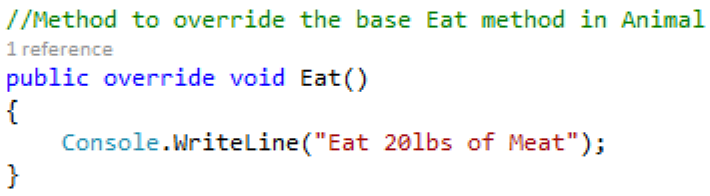
Step 4: Override the `MakeNoise`() method in each subclass to perform the correct animal noise e.g. Wolf howls.

Step 5: We will now override the `Eat`() method in the base class for the Tiger subclass to allow the tiger to eat the correct amound of meat. First Mark the `Eat`() method as virtual in the base Animal class:



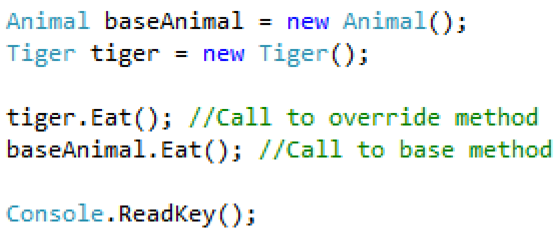
```
//Virtual method to allow overriding in subclasses
1 reference
public virtual void Eat()
{
    Console.WriteLine("Eat");
}
```

Next create a `Eat`() method in the Tiger class and include the **`override`** keyword in the method declaration as shown below to override the base method:

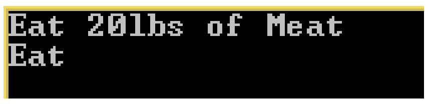


```
//Method to override the base Eat method in Animal
1 reference
public override void Eat()
{
    Console.WriteLine("Eat 20lbs of Meat");
}
```

Step 6: Call the `Eat`() method on a Tiger object and an Animal object to view which method is executed as shown below:



```
Animal baseAnimal = new Animal();
Tiger tiger = new Tiger();

tiger.Eat(); //Call to override method
baseAnimal.Eat(); //Call to base method

Console.ReadKey();
```



```
Eat 20lbs of Meat
Eat
```

Step 7: Override the `Eat`() method in each subclass to allow each animal to eat the correct food and the correct quantity e.g. Wolf eat 10lbs of Meat.

Step 8: Time to be creative so decide on a new method that the animal will perform. Add a new method in the Animal class that can be overridden and then override the method in the subclasses of the animals that can perform this action.

Example: Hippos can swim and love it but Wolves do not like to.

TASK 4: Multiple inheritance

So Zoo Park is starting to take shape but could we make our inheritance tree better. At the moment Lion and Tiger inherit from the base class Animal but since they are both Big Cats they will have some common methods and attributes that are common to cats but not common to other animals e.g. birds.

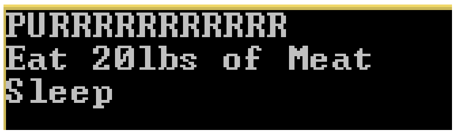
Why don't we make a Feline class that inherits from Animal and make the Lion and Tiger inherit from the Feline class instead of the Animal class which will contain all of the common methods and attributes specific to Cats and also all from the Animal base class?

Step 1: Add the New `Feline` class that inherits from the base Animal class:

```
Animal baseAnimal = new Animal(); //Base Class
Feline feline = new Feline(); //Inherits from Animal
```

Step 2: Add the common Feline attributes and methods to the feline class e.g. `Purr`() and call to a method of the Tiger class, the Feline class and the base Animal class as shown below:

```
Animal baseAnimal = new Animal(); //Base Class
Feline feline = new Feline(); //Inherits from Animal
Tiger tiger = new Tiger(); //Inherits from Feline and Animal

tiger.Purr(); //Call to Purr method in Feline Class
tiger.Eat(); //Call to override method in Tiger Class
tiger.Sleep(); //Call to base method
```



```
PURRRRRRRRRRR
Eat 20lbs of Meat
Sleep
```

As you can see, the `Purr`() method is executed in the Feline class, the `Eat`() override method is executed in the Tiger class and the `Sleep`() method is executed in the base Animal class.

Step 3: The Zoo has a new arrival and it is a Penguin. Now add a new `Bird` class that inherits from the base Animal class as there are now two birds that will share some common attributes and methods e.g. wingspan and bird species.

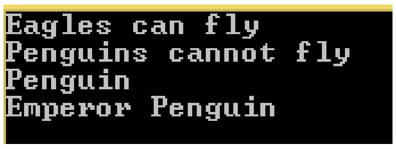
Now add a new `Penguin` class that uses the Bird class and change the Eagle class to now inherit from the Bird class instead of directly to the Animal base class.

Think of methods and attributes that only birds would have and methods and attributes that are unique to each type of bird.

For example, you could add a Fly method in to the Bird class and in each subclass state if that bird can fly or not by overriding the method.

Example code is shown on next page:

```
Eagle eagle = new Eagle();
Penguin penguin = new Penguin();

eagle.Fly(); //Call to override method in Eagle Class
penguin.Fly(); //Call to override method in Penguin Class

penguin.BirdSpecies = "Penguin"; //Assigning variable in Bird class
penguin.PenguinSpecies = "Emperor Penguin";  //Assigning variable in Penguin class

Console.WriteLine(penguin.BirdSpecies); //Output BirdSpecies from Bird class
Console.WriteLine(penguin.PenguinSpecies); //Output PenguinSpecies from Penguin class
```



```
Eagles can fly
Penguins cannot fly
Penguin
Emperor Penguin
```

TASK 5: WHAT'S THE OUTPUT?

You should try to work this out on paper and then transfer the code into Visual Studio and run it to see what output you actually get. Remember that if you don't get the answer you expect to check both your workings on paper and the code.

```
class A
{
    0 references
    public virtual void m1()
    {
        Console.WriteLine("A's M1");
    }

    0 references
    public void m2()
    {
        Console.WriteLine("A's M2");
    }
}
```

```
class B : A
{
    1 reference
    public override void m1()
    {
        Console.WriteLine("B's M1");
    }

    0 references
    public void m2()
    {
        Console.WriteLine("B's M2");
    }
}
```

```
class C: B
{
    0 references
    public void m1()
    {
        Console.WriteLine("C's M1");
    }
}
```

a)

```
A a = new A();
B b = new B();
C c = new C();

a.m1();
b.m1();
```

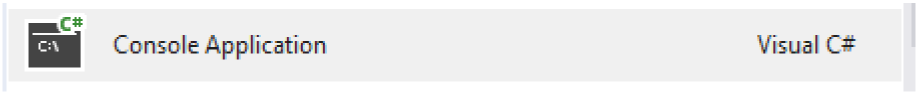
b)

```
A a = new A();
B b = new B();
C c = new C();

a.m2();
b.m2();
c.m1();
c.m2();
```

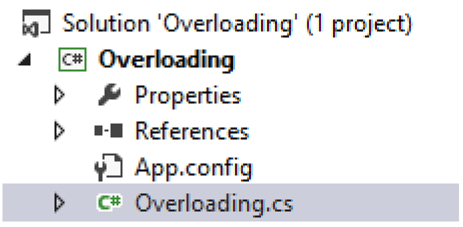
TASK 6: Overloading

`Overloading` is what happens when you have two methods with the same name but different signatures.

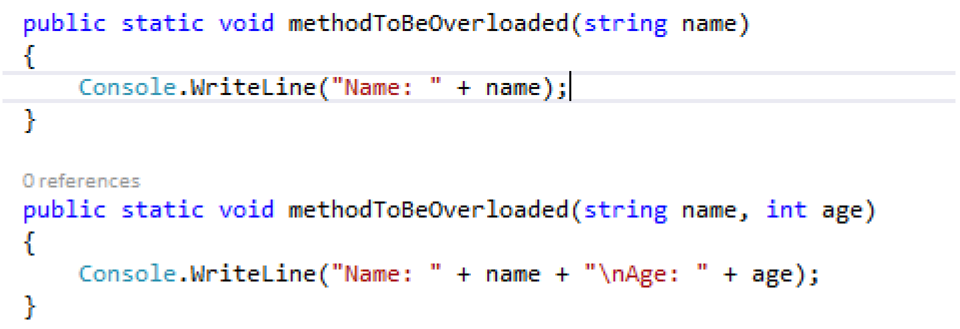
Step 1: Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Console Application and name it `e.g. Overloading`


Console Application Visual C#

Step 2: Right Click on Program.cs in the Solution explorer and rename to Overloading.cs and select Yes in the followed popup:

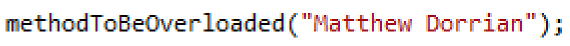

Solution 'Overloading' (1 project)
Overloading
Properties
References
App.config
Overloading.cs

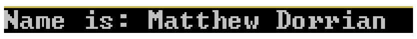
Step 3: Now we will add two methods that have the same name but the parameter list in the method signature is different. This concept is `Overloading`:

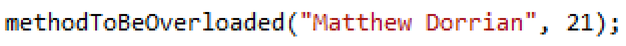

```
public static void methodToBeOverloaded(string name)
{
    Console.WriteLine("Name: " + name);
}

0 references
public static void methodToBeOverloaded(string name, int age)
{
    Console.WriteLine("Name: " + name + "\nAge: " + age);
}
```

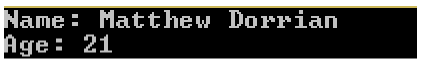
We now have two methods that can be called by passing in one variable or two and print out the variables to the Console.

Step 4: Call each method using different parameters and view the Console:


```
methodToBeOverloaded("Matthew Dorrian");
```


```
Name is: Matthew Dorrian
```


```
methodToBeOverloaded("Matthew Dorrian", 21);
```


```
Name: Matthew Dorrian
Age: 21
```

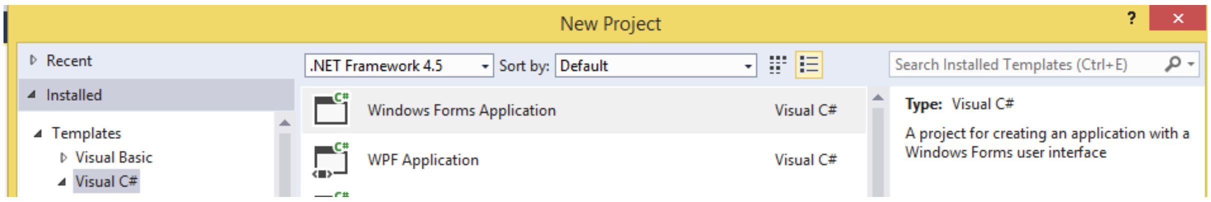
At *compile time*, the compiler works out which one it's going to call, based on the compile time types of the arguments and the target of the method call e.g. if the method call contains a string and an int, call the method that prints both.

EXERCISE 2

The tasks in the second part of this section will use Windows Forms Applications to demonstrate inheritance and to further enhance the use of GUI's.

TASK 7: Sandwich Ordering

Step 1: Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. SandwichOrdering`.

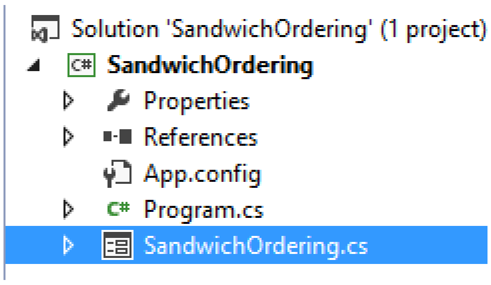


Windows Forms Application	Visual C#
WPF Application	Visual C#

Type: Visual C#

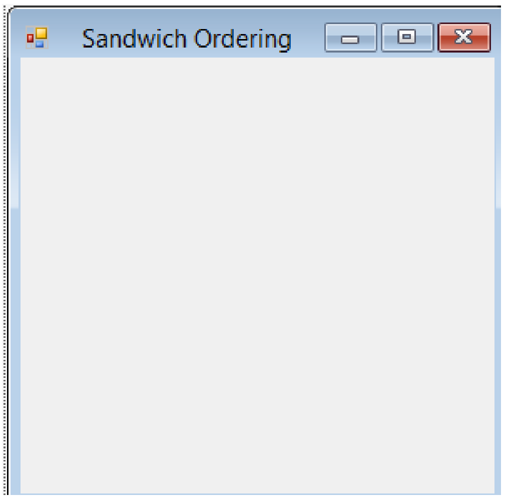
A project for creating an application with a Windows Forms user interface

Step 2: Right Click on Form1.cs in the Solution explorer and rename to SandwichOrdering.cs:

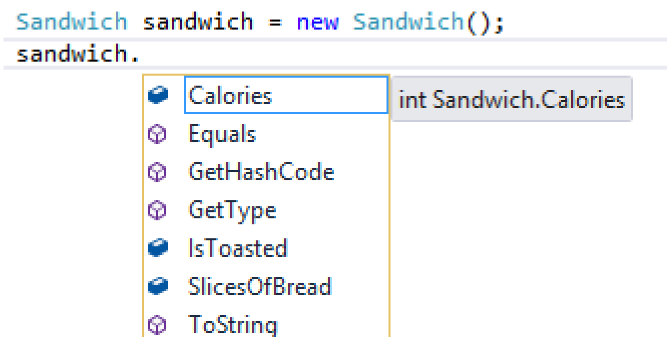


Solution 'SandwichOrdering' (1 project)
- SandwichOrdering
  - Properties
  - References
  - App.config
  - Program.cs
  - SandwichOrdering.cs

Step 3: Using the properties window, change the title of your form to something meaningful e.g. Sandwich ordering:

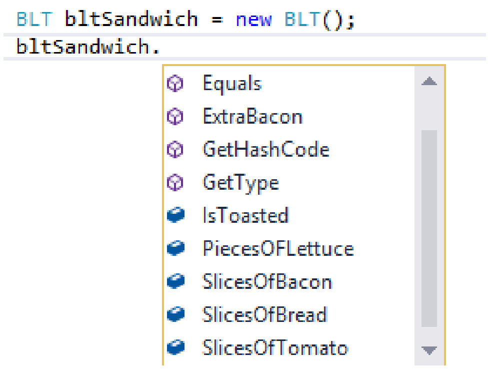


Step 4: We want to implement a solution that uses a base class and a more specific sandwich class for a particular sandwich e.g. BLT. Firstly create a `Sandwich` base class that will contain the common attributes and methods e.g. `IsToasted, Calories`:



```
Sandwich sandwich = new Sandwich();
sandwich.
```

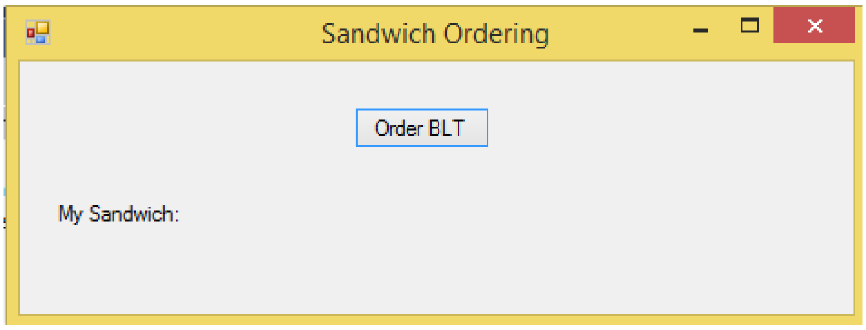
Step 5: Next we want to create a subclass of Sandwich called `BLT`. This subclass will inherit all the public methods and attributes from Sandwich but will have its own specific attributes and methods e.g. `SlicesOfBacon`, `SlicesOfTomato`, `AmountOfLettuce`, `ExtraBacon`():



```
BLT bltSandwich = new BLT();
bltSandwich.
```

As you can see, BLT has all the methods and attributes of the Sandwich base class and its own methods and attributes.

Step 6: Create the following GUI using the toolbox to add elements to the form e.g. Labels, Buttons. You can reposition any element on the screen by clicking and dragging the element to the desired location:

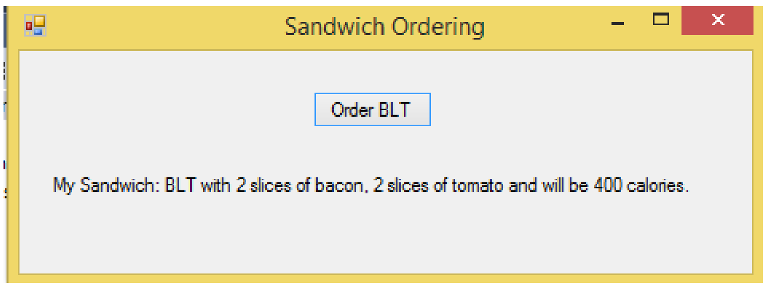


On the button click, we want to populate the My Sandwich label with details of a BLT sandwich that has 2 slices of bacon, 2 slices of tomato and 4 pieces of Lettuce and this sandwich will be 400 Calories.

Step 7: Double click on the Order Button (Rename button first to meaningful name using properties window i.e. btnBTL) to create the click event method automatically:

```
private void btnBLT_Click(object sender, EventArgs e)
{
    //Generated click event method
}
```

Step 8: Add code to the click event method to create a BLT object and populate it using the public attributes e.g. `blt.SlicesOfBacon = 2`. The finished form should look like the following:

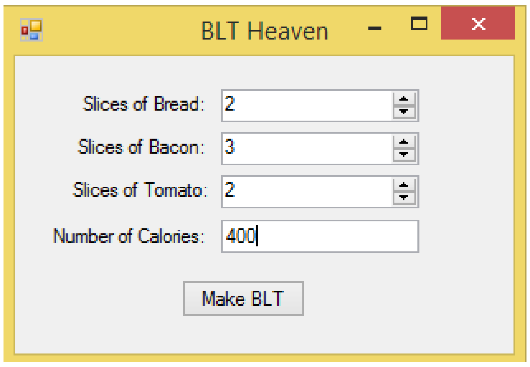


My Sandwich: BLT with 2 slices of bacon, 2 slices of tomato and will be 400 calories.

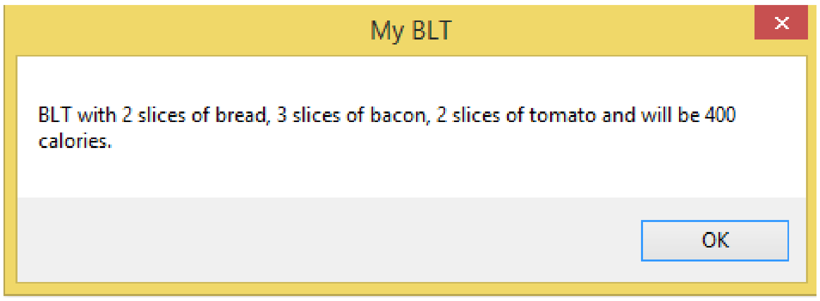
(Hint: Create instance of BLT object and assign each variable with the required values e.g. 2 bacon slices)

TASK 8: Sandwich Ordering with Input

Create a new project called `SandwichOrderingImproved`. Create a form that allows the User to enter in the amount of each ingredient for their BLT e.g. slices of bacon and output the BLT in a `MessageBox`. You will have to create the `Sandwich` and `BLT` class again. An example layout has been show below but don't be afraid to make your own:



Slices of Bread: 2
Slices of Bacon: 3
Slices of Tomato: 2
Number of Calories: 400



BLT with 2 slices of bread, 3 slices of bacon, 2 slices of tomato and will be 400 calories.

Extra: Now update the GUI to allow the user to specify if the sandwich is toasted or not.

TASK 9: CAN YOU SPOT THE ERROR?

a)
```
public class Coca Cola ; FizzyDrink
{
}
```

b)
```
Tiger tony = new Tiger();
Tiger.TigerSpecies = “Bengal Tiger”
```

c)
```
public class FizzyDrink
{
        public void virtual TakeDrink()
        {
        }
}
```

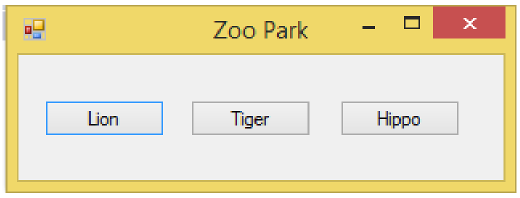
d)
```
public class Bear : Animal
{
        public overridden void Eat()
        {
        }
}
```

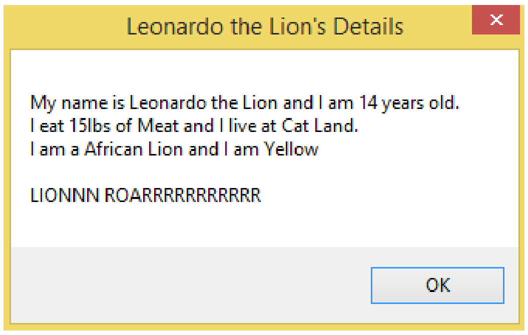
e)
```
BLT sandwich = new BLT();
new sandwich.SlicesOfBacon = 5;
```

f)
```
Bear bear = new Bear();
MessageBox.Show(“The Bear’s name is “ bear.Name + “ /n and he is ” +
bear.Height + “ feet tall);
```

EXERCISE 3

TASK 10: Create a GUI version of Zoo park which will show the details of a Tiger, a Lion and a Hippo in a `MessageBox` on a button click. Remember to create the base `Animal` class, the subclasses for the animals and if two animals are say Cats, create an in-between `Feline` class. Example UI is shown below but feel free to experiment with the UI:

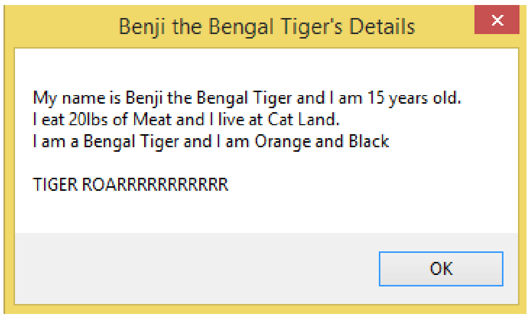




Leonardo the Lion's Details

My name is Leonardo the Lion and I am 14 years old.
I eat 15lbs of Meat and I live at Cat Land.
I am a African Lion and I am Yellow

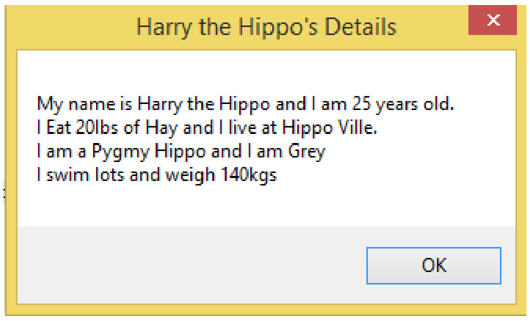
LIONNN ROARRRRRRRRRRR



Benji the Bengal Tiger's Details

My name is Benji the Bengal Tiger and I am 15 years old.
I eat 20lbs of Meat and I live at Cat Land.
I am a Bengal Tiger and I am Orange and Black

TIGER ROARRRRRRRRRRR



Harry the Hippo's Details

My name is Harry the Hippo and I am 25 years old.
I Eat 20lbs of Hay and I live at Hippo Ville.
I am a Pygmy Hippo and I am Grey
I swim lots and weigh 140kgs

If you feel like you want to add more functionality, add a menu bar which shows you stats on the Zoo in a Message Dialog e.g. Number of each animal and add more animal subclasses e.g. Seal.