# Classes, Objects and Data Encapsulation

### What is an Object?

Objects are the fundamental element of object-oriented programming. Look around right now and you'll find many examples of real-world objects: your dog, your computer, your bicycle.

Real-world objects share two characteristics: They all have **state** and **behaviour**. Dogs have state (name, color, breed, hungry) and behaviour (barking, fetching, wagging tail). Bicycles also have state (gear, pedal cadence, speed) and behaviour (changing gear, changing pedal cadence, applying brakes). Identifying the state and behaviour for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Take a minute right now to observe the real-world objects that are in your immediate area. For each object that you see, ask yourself two questions: "What possible states can this object be in?" and "What possible behaviour can this object perform?". Make sure to write down your observations. As you do, you'll notice that real-world objects vary in complexity; your desktop lamp may have only two possible states (on and off) and two possible behaviours (turn on, turn off), but your desktop radio might have additional states (on, off, current volume, current station) and behaviour (turn on, turn off, increase volume, decrease volume, seek, scan, and tune). You may also notice that some objects, in turn, will also contain other objects. These real-world observations all translate into the world of object-oriented programming.

Software ***objects*** are conceptually similar to real-world objects: they too consist of state and related behaviour. An object stores its state in ***fields*** and exposes its behaviour through ***methods***. Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as ***data encapsulation*** — a fundamental principle of object-oriented programming.

### What is a Class?

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created.

> ***A class is a blueprint for an object.***

What this basically means is that we provide a blueprint, or an outline of an object. This blueprint is valid whether we have one or one thousand such objects. A class does not represent an object; it represents all the information a typical object should have as well as all the methods it should have.

---

In today's lesson, we will:

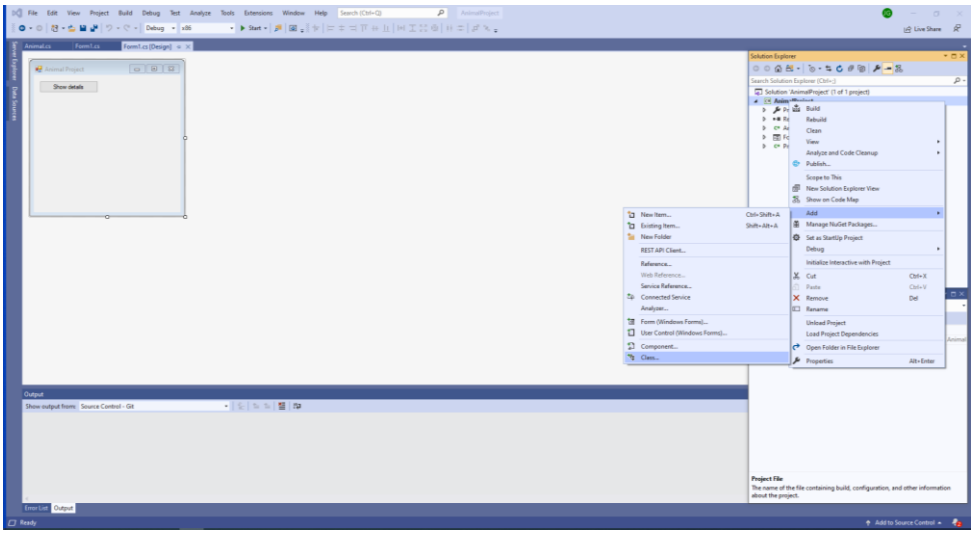
- Declare a class,
- Create objects that belong to that class, and
- Use the objects by calling its methods.

## Task 1: The Animal Project

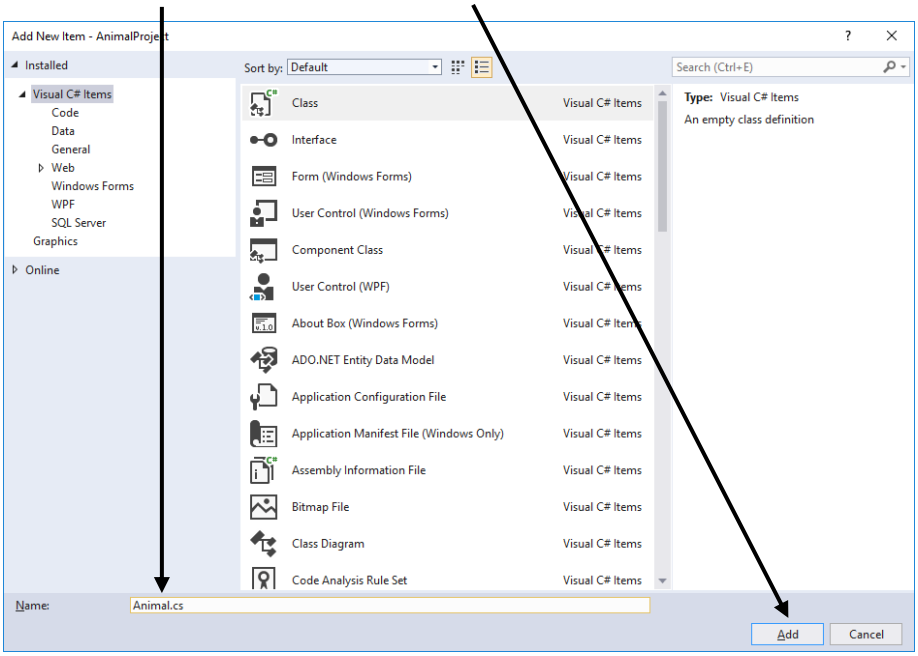
1. Create a new application. Call the project **AnimalProject**.

2. Add a **button** to the Form. Change the Text property to *Show Details*.

3. In the **Solution Explorer** window, right click on the **AnimalProject** namespace, select **Add/Class.**



4. Name the class **Animal.cs** and select **Add**.



Note the name of a class always begins with a capital letter. Make sure that your class has a name that is different from the namespace.

5. Create the **Animal Class**.

In a **UML Class Diagram**, the Animal class is described as:

| Animal |
| --- |
| type : string<br>name : string<br>legs : int |
| Animal (type : string, name: string, legs : int)<br>ListDetails: string |

Note there are three parts to a class diagram.

The first part is the **name** of the class.

The second part contains the **fields** which describe the **state** of the class. Note fields always begin with a lower case letter and are always private so that they are limited to their class. Each field is given a name and a datatype: type and name are Strings and legs is an integer.

The third part contains **methods** that describe the **behaviours** of the class.
The **Constructor** method is called **Animal**. It has exactly the same name as the class, and also starts with a capital letter. It is a special Method that has **no return type**. The constructor provides the initial values for all the fields.

Add another **Method**, ListDetails that returns a string.

Select the **Animal.cs** tab on the main window. This contains the code for the Animal class. Add **comments** as you write your code.

```
/*  The Animal Class
    contains the fields, constructor and methods that define an Animal
*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AnimalProject
{
    public class Animal
    {
        // fields, also called private data members
        private string type;
        private string name;
        private int legs;

        // constructor, used to assign initial values to the fields
        public Animal()
        {
        }

        // method, user defined
        public string ListDetails()
        {
        }
    }
}
```

6. RightClick in the AnimalProject namespace, select **View/View Class Diagram**. These class diagrams are automatically created as you build the code for your class. Alternatively you can add fields and methods here that will also populate your code.

7. Returning to the **Animal.cs** file, go to the Animal **constructor**. In the constructor, we initialize all fields. Here the values for the fields to be used are passed in as a list of parameters.

```
public Animal(string type, string name, int legs)
{
     this.type = type;
     this.name = name;
     this.legs = legs;
}
```

We use the same name for the parameter as the field! This may seem confusing but it also links the two together for meaning. We distinguish the field from the parameter by adding the keyword **this** before the field name, meaning it belongs to the Form1. It is the only time you use **this**.

8. The ListDetails() **method** returns a string with the details about an individual animal.. Note a method always starts with a verb and a capital letter. This method is public because we call it from outside the class.

```
public string ListDetails()
{
     return ("I am a " + type + ", my name is " + name + " and I have "
       + legs + " legs");
}
```

9. **Create 2 Animal Objects**

In the *Form1* class, remember to always start with **the block comment**!

```
/*  Program name:
    Project file name:
    Author:
    Date:
    Language:            C#
    Platform:            Microsoft Visual Studio 2019
    Purpose:
    Description:
    Known Bugs:
    Additional Features:
*/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

Objects are declared in Form1, as private data members. I want to have 2 Animals, one called dog and the other called cat:

```
namespace AnimalProject
{
        public partial class Form1 : Form
        {
             private Animal dog;
             private Animal cat;
```

To create the animals, from the Form's constructor, call the Animal constructor using the **new** operator. This is where we provide the arguments, the actual data values.

```
public Form1()
        {
             InitializeComponent();

             dog = new Animal("dog", "Fido", 3);
             cat = new Animal("cat", "Fluffy", 4);
        }
```

10. **Use your Objects by calling their Methods**

Two animals have been created but at the moment they don't do anything.
On the Form, add a button with Text Property "Show details"
Add the Event Handler for the button1_Click event.

```
private void button1_Click(object sender, EventArgs e)
        {
             MessageBox.Show(dog.ListDetails());
             MessageBox.Show(cat.ListDetails());
        }
```

---

**Task 2: Clowns**

1. Create a new application. Call the project **Circus**.

2. Add a class called **Clown**. Note a class always begins with a capital letter.

   Add the **Fields**: name of type string, height of type integer and age of type integer. Note fields always begin with a lower case letter.

   Then add the **Constructor** method called Clown. It is exactly the same name as the class, and also starts with a capital letter. It is a special Method that has no return type.

   Add another **Method**, TalkAboutYourself that returns a String.

```
/*  The Clown Class
    contains the fields, constructor and methods that define a clown
*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Circus
{
    public class Clown
    {
        // fields, also called private data members
        private string name;
        private int height;
        private int age;

        // constructor, used to assign initial values to the fields
        public Clown()
        {
        }

        // method, user defined
        public string TalkAboutYourself()
        {
        }
    }
}
```

In the **Constructor**, pass in the parameter values needed to initilalise the fields.

```
public Clown(string name, int height, int age)
        {
            this.name = name;
            this.height = height;
            this.age = age;
        }
```

Notice that we use the same name for the parameter and for the field. The keyword  this is used to distinguish the field from the local variable name.

The TalkAboutYourself() **method** returns a string with the details about a clown. Note a method always starts with a verb and a capital letter. Methods need to be public because we call them from outside the class.

```
public string TalkAboutYourself()
        {
            return("My name is " + name + ". I am " + age + " years old. And I
              am " + height + " centimeters tall.");
        }
```

3. **Create 3 Clown Objects**

Remember to always start with **the block comment!**

Objects are declared in the form, as private data members. I want to have 3 clowns:

```
/*  Program name:
    Project file name:
    Author:
    Date:
    Language:            C#
    Platform:            Microsoft Visual Studio 2017
    Purpose:
    Description:
    Known Bugs:
    Additional Features:
*/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Circus
{
    public partial class Form1 : Form
    {
        private Clown clown1;
        private Clown clown2;
        private Clown clown3;
```

To create the clowns, from the Form's constructor, call the Clown constructor using the **new** operator. This is where we provide the arguments, the actual data values.

```
public Form1()
        {
            InitializeComponent();

            clown1 = new Clown("Crusty", 160, 45);
            clown2 = new Clown("Charlie", 100, 18);
            clown3 = new Clown("Clary", 180, 67);
        }
```

4. **Use your Objects by calling their Methods**

Three clowns have been created but at the moment we cannot use them.
On the Form, add a button with Text Property "Which clowns are available today?"
Add the Event Handler for the button1_Click event.

```
private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(clown1.TalkAboutYourself());
            ……..
        }
```

**Task 3: Improve your Circus application**

1. Put the 3 clowns into an array.

2. Add a PictureBox to the Form. Add the code so that when the user clicks the button, the appropriate associated image is displayed.

3. Add all images to the Resources folder. Check in the Solution Explorer.

---

**Task 4: Create a Polytech application**

1. Create a Student class that has 4 fields (FamilyName, GivenName, Age and StudentNumber), a constructor and 2 methods (Enrol and Withdraw). When you enrol a student, all their data is displayed in a sentence in a MessageBox. When a student is withdrawn, only their names and student number is in the sentence displayed in a MessageBox.

2. Create 1 student object in the Form1.cs. Add 2 buttons to the form. Add the code so that when the user clicks a button, the appropriate sentence is displayed in the MessageBox.

---

**Task 5: Create a WorkPlace application**

1. Create an Employee class that has 4 fields (FamilyName, GivenName, Position and EmployeeNumber), a constructor, and a ListDetails method. In the ListDetails method, all data concerning that employee is returned as a String.

2. Create 5 employee objects in the Form1.cs. Add a button to the form. Add the code so that when the user clicks the button, all employee data is written to a ListBox.

---