Object Oriented Programming - 1

A5 – Principles Applied Program to Design

What is Object Oriented Programming ?

- Procedural programming is about writing procedures or methods that perform operations on the data.

  COBOL, BASIC, PASCAL, FORTRAN, C

- Object-oriented programming is about creating objects that contain both data and methods.

  C++, C#, JAVA, Python

- Object-oriented programming provides concepts that help modelling complicated systems of real world into manageable software solutions.

What is Object Oriented Programming ?

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time.

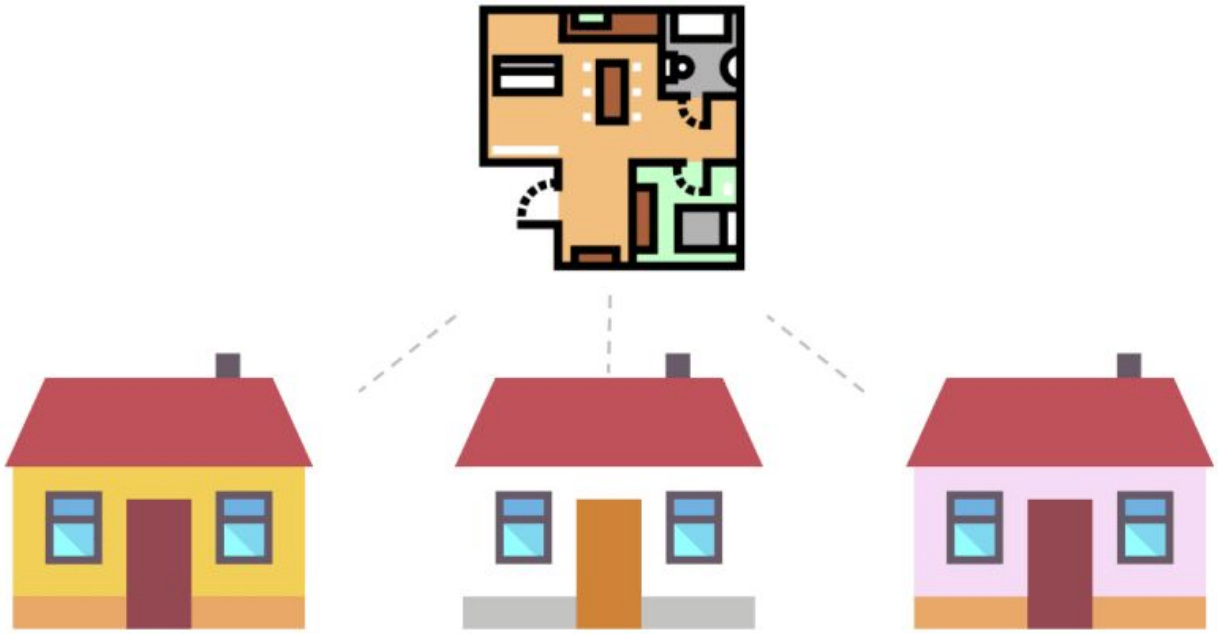
What are Classes and Objects ?

Classes and objects are the two main aspects of object-oriented programming:

- A class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit their structure from the class

What are Classes and Objects ?

A class is a blueprint for creating an object. It is similar to the blue print of a house.



What are Classes and Objects ?

A class defines attributes and behavior.

If we are to model a car in our application then we could define :

- Attributes of the car like, model, fuel, make
- Behaviors like start, break, accelerate etc.

```
public class Car{
    private string _color;
    private string _model;
    private string _makeYear;
    private string _fuelType;

    public void Start(){
        ..
    }

    public void Stop(){
        ..
    }

    public void Accelerate(){
        ..
    }
}
```

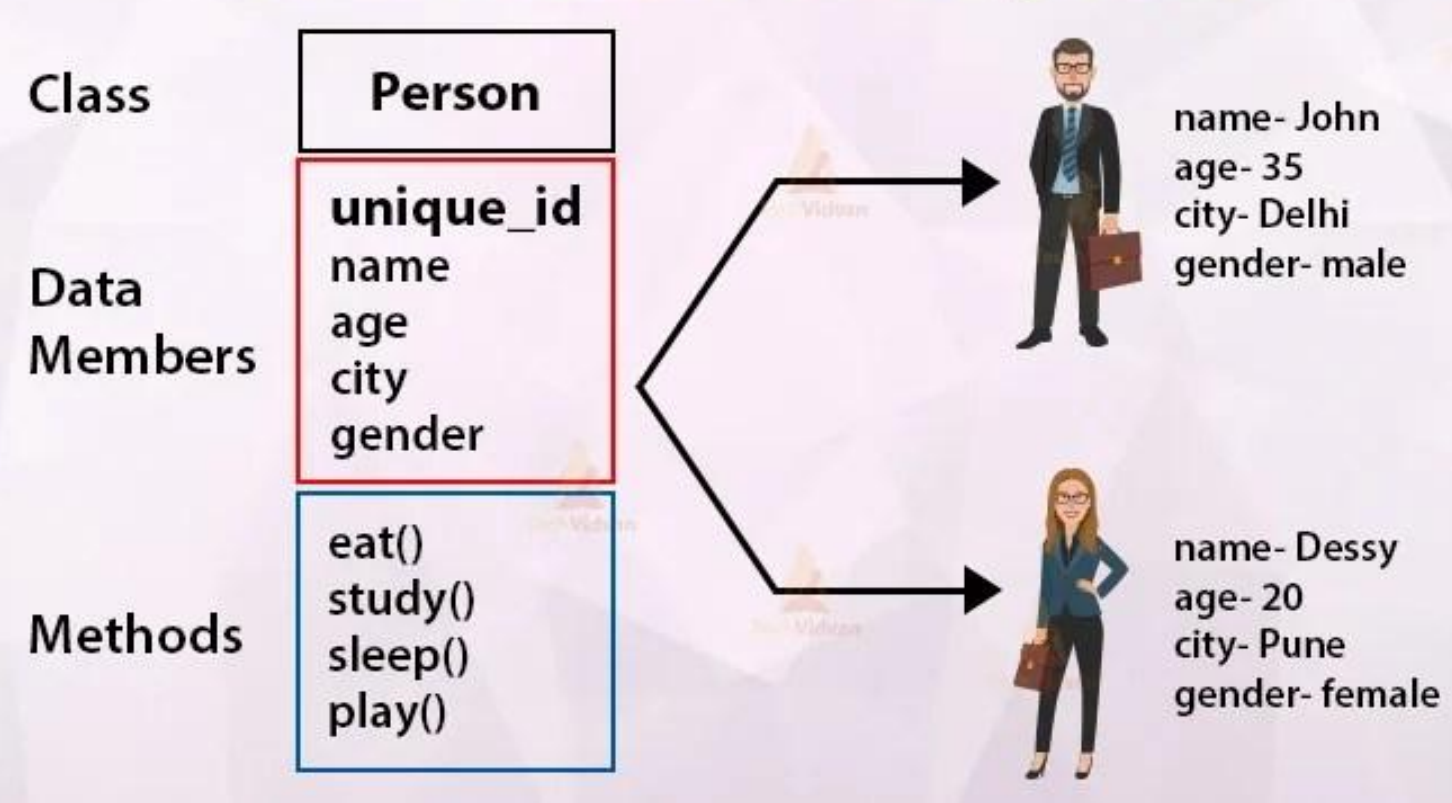
What are Classes and Objects ?

Using the same class "Car" we can create different objects having variation in model, fuel type and make year while having the same common behavior.

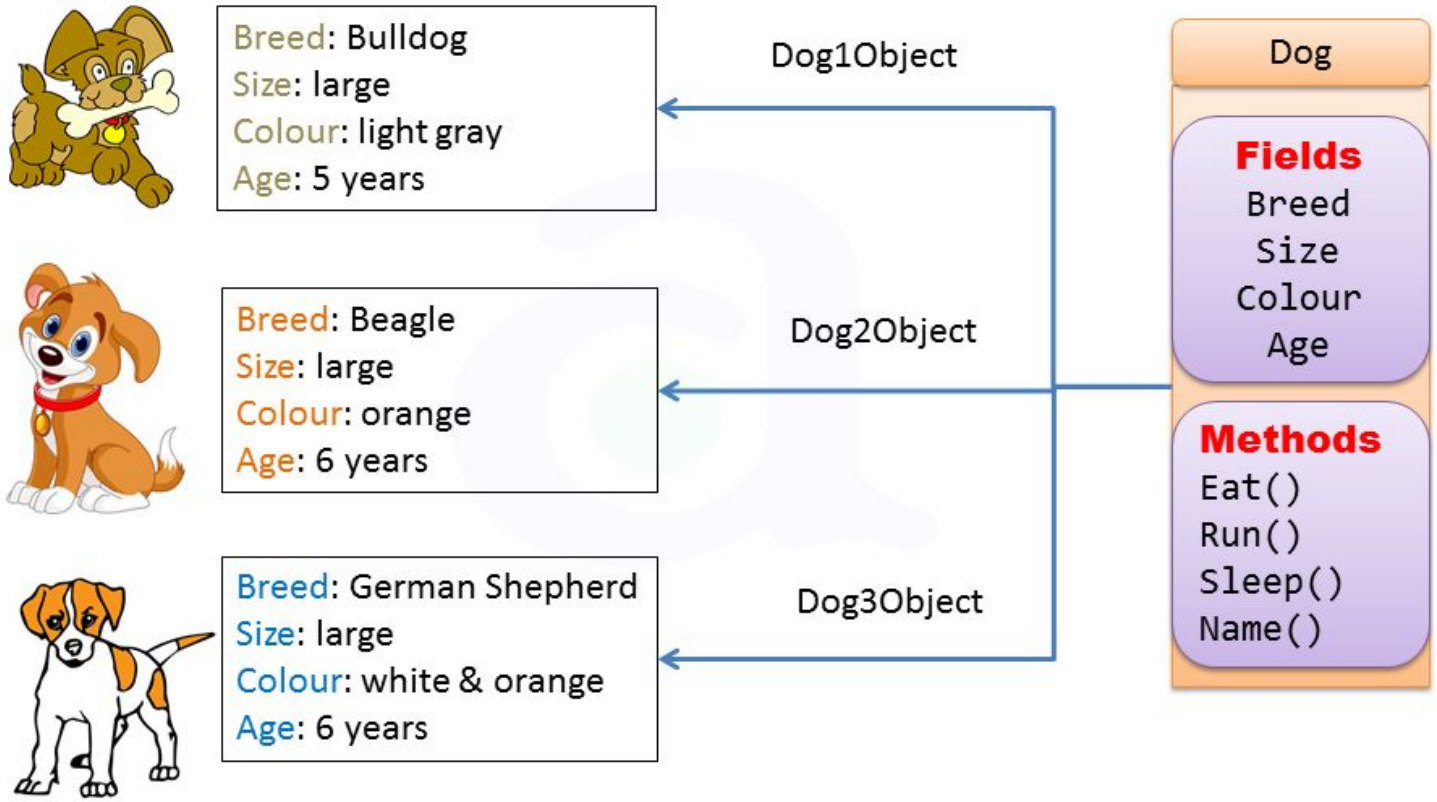
Object 1	
Model	Volkswagen Polo
Fuel	Petrol
Make	2017
Start() Break() Accelerate()	

Object 2	
Model	Volkswagen Vento
Fuel	Diesel
Make	2017
Start() Break() Accelerate()	

What are Classes and Objects ?



What are Classes and Objects ?



Create a Class

Everything in C# is associated with classes and objects, along with its attributes and methods.

- In real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

Create a class named "Car" with a variable color:

```
class Car
{
  string color = "red";
}
```

Create a Class

- When a variable is declared directly in a class, it is often referred to as a field (or attribute).
- It is not required, but it is a good practice to start with an uppercase first letter when naming classes.

Create a class named "Car" with a variable color:

```
class Car
{
  string color = "red";
}
```

Also, it is common that the name of the C# file and the class matches, as it makes our code organized. However it is not required (like in Java).

Create an Object

An object is created from a class. We have already created the class named Car, so now we can use this to create objects.

To create an object of Car, specify the class name, followed by the object name, and use the keyword `new`:

Example

Create an object called "myObj" and use it to print the value of color:

```
class Car
{
  string color = "red";

  static void Main(string[] args)
  {
    Car myObj = new Car();
    Console.WriteLine(myObj.color);
  }
}
```

Multiple Objects

You can create multiple objects of one class:

Example

Create two objects of `Car` :

```
class Car
{
  string color = "red";
  static void Main(string[] args)
  {
    Car myObj1 = new Car();
    Car myObj2 = new Car();
    Console.WriteLine(myObj1.color);
    Console.WriteLine(myObj2.color);
  }
}
```

Using Multiple Classes

You can also create an object of a class and access it in another class.

This is often used for better organization of classes (one class has all the fields and methods, while the other class holds the `Main()` method (code to be executed)).

Using Multiple Classes

Did you notice the `public` keyword? It is called an **access modifier**, which specifies that the `color` variable/field of `Car` is accessible for other classes as well, such as `Program`.

prog2.cs

```
class Car
{
  public string color = "red";
}
```

prog.cs

```
class Program
{
  static void Main(string[] args)
  {
    Car myObj = new Car();
    Console.WriteLine(myObj.color);
  }
}
```

Class Members

Fields and methods inside classes are often referred to as "Class Members":

Example

Create a `Car` class with three class members: **two fields** and **one method**.

```
// The class
class MyClass
{
  // Class members
  string color = "red";         // field
  int maxSpeed = 200;           // field
  public void fullThrottle()    // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
```

Class Members & Fields

- In the previous slides, you learned that variables inside a class are called fields, and that you can access them by creating an object of the class, and by using the dot syntax (.).
- The following example will create an object of the `Car` class, with the name `myObj`. Then we print the value of the fields `color` and `maxSpeed`:

Class Members & Fields

Example

```
class Car
{
  string color = "red";
  int maxSpeed = 200;

  static void Main(string[] args)
  {
    Car myObj = new Car();
    Console.WriteLine(myObj.color);
    Console.WriteLine(myObj.maxSpeed);
  }
}
```

Class Members & Fields

You can also leave the fields blank, and modify them when creating the object:

Example

```
class Car
{
  string color;
  int maxSpeed;

  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.color = "red";
    myObj.maxSpeed = 200;
    Console.WriteLine(myObj.color);
    Console.WriteLine(myObj.maxSpeed);
  }
}
```

Class Members & Fields

This is especially useful when creating multiple objects of one class:

Example

```
class Car
{
  string model;
  string color;
  int year;

  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;

    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Object Methods

- Methods normally belongs to a class, and they define how an object of a class behaves.
- Just like with fields, you can access methods with the dot syntax.
- However, note that the method must be `public`. And remember that we use the name of the method followed by two parantheses `()` and a semicolon `;` to call (execute) the method:

Object Methods

Example

```
class Car
{
  string color;                // field
  int maxSpeed;                // field
  public void fullThrottle()   // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }

  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.fullThrottle();  // Call the method
  }
}
```

Object Methods

- Why did we declare the method as `public`, and not `static`?
- The reason is simple: a `static` method can be accessed without creating an object of the class, while `public` methods can only be accessed by objects.

Example

```
class Car
{
  string color;                // field
  int maxSpeed;                // field
  public void fullThrottle()   // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }

  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.fullThrottle();  // Call the method
  }
}
```

Use Multiple Classes

Remember from previous slides, that we can use multiple classes for better organization (one for fields and methods, and another one for execution). This is recommended:

prog2.cs

```
class Car
{
  public string model;
  public string color;
  public int year;
  public void fullThrottle()
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
```

prog.cs

```
class Program
{
  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;

    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Use Multiple Classes

The `public` keyword is called an **access modifier**, which specifies that the fields of `Car` are accessible for other classes as well, such as `Program`.

prog2.cs

```
class Car
{
  public string model;
  public string color;
  public int year;
  public void fullThrottle()
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
```

prog.cs

```
class Program
{
  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;

    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Constructor

A constructor is a **special method** that is used to initialize objects. The advantage of a constructor, is that it is called when an object of a class is created. It can be used to set initial values for fields:

Example

Create a constructor:

```
// Create a Car class
class Car
{
  public string model;  // Create a field

  // Create a class constructor for the Car class
  public Car()
  {
    model = "Mustang"; // Set the initial value for model
  }

  static void Main(string[] args)
  {
    Car Ford = new Car();  // Create an object of the Car Class (this will call the co
    Console.WriteLine(Ford.model);  // Print the value of model
  }
}

// Outputs "Mustang"
```

Constructor

- Note that the constructor name must **match the class name**, and it cannot have a **return type** (like `void` or `int`).
- Also note that the constructor is called when the object is created.
- All classes have constructors by default: if you do not create a class constructor yourself, C# creates one for you.

  However, then you are not able to set initial values for fields.

Example

Create a constructor:

```
// Create a Car class
class Car
{
  public string model;  // Create a field

  // Create a class constructor for the Car cl
  public Car()
  {
    model = "Mustang"; // Set the initial valu
  }

  static void Main(string[] args)
  {
    Car Ford = new Car();  // Create an object
    Console.WriteLine(Ford.model);  // Print t
  }
}

// Outputs "Mustang"
```

Constructor Parameters

Constructors can also take parameters, which is used to initialize fields.

The following example adds a `string modelName` parameter to the constructor. Inside the constructor we set `model` to `modelName` (`model=modelName`).

When we call the constructor, we pass a parameter to the constructor ("`Mustang`"), which will set the value of `model` to "`Mustang`":

Example

```
class Car
{
  public string model;

  // Create a class constructor with a pa
  public Car(string modelName)
  {
    model = modelName;
  }

  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang");
    Console.WriteLine(Ford.model);
  }
}

// Outputs "Mustang"
```

Constructor Parameters

You can have as many parameters as you want:

Example

```
class Car
{
  public string model;
  public string color;
  public int year;

  // Create a class constructor with multiple parameters
  public Car(string modelName, string modelColor, int modelYear)
  {
    model = modelName;
    color = modelColor;
    year = modelYear;
  }

  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang", "Red", 1969);
    Console.WriteLine(Ford.color + " " + Ford.year + " " + Ford.model);
  }
}


// Outputs Red 1969 Mustang
```

Next lecture

- In the next lecture we continue focusing on «Principles of Logic Applied to Program Desingn»
- “Microsoft Visual C# Step by Step” Microsoft Press