OOP LAB 7 (01 Nov 22)

C# Object Oriented Programming
By: Miss Tia

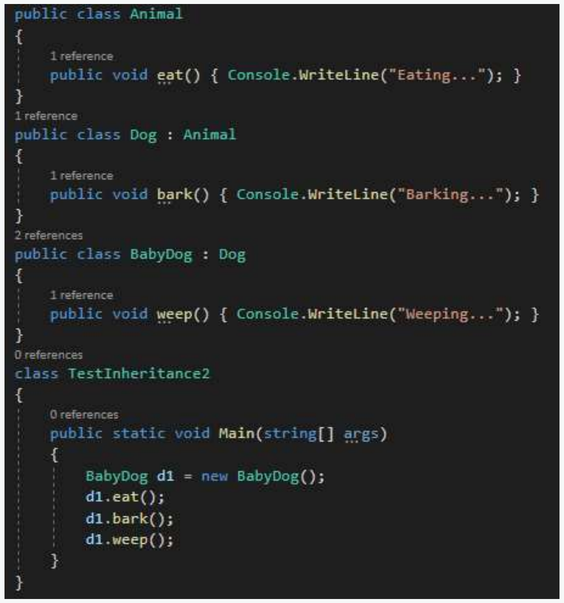
Polymorphism and Overriding Methods

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

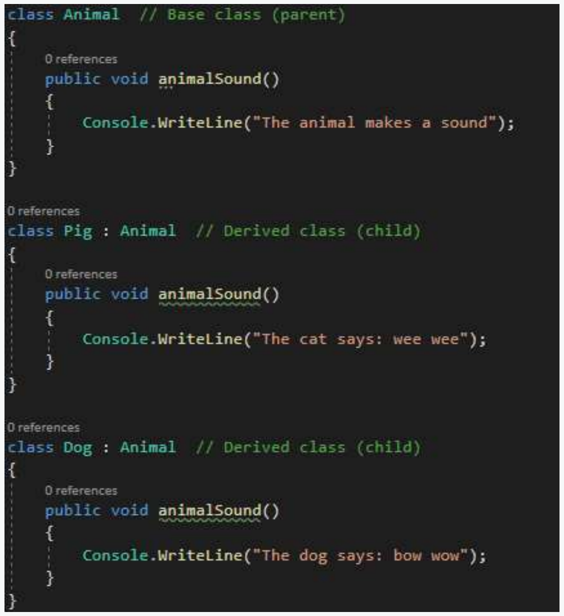
Inheritance lets us inherit fields and methods from another class. Polymorphism uses those methods to perform different tasks.

This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the cat meows, dog barks and bird chirps):



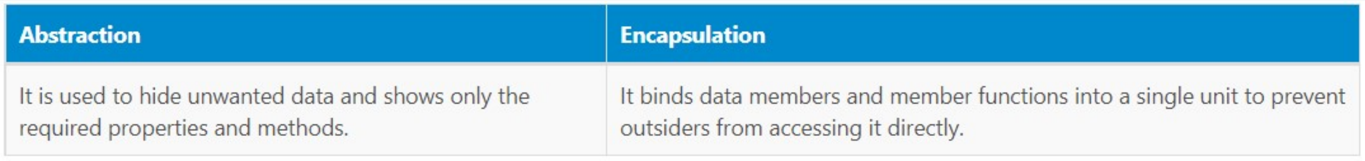
```
public class Animal
{
    1 reference
    public void eat() { Console.WriteLine("Eating..."); }
}
1 reference
public class Dog : Animal
{
    1 reference
    public void bark() { Console.WriteLine("Barking..."); }
}
2 references
public class BabyDog : Dog
{
    1 reference
    public void weep() { Console.WriteLine("Weeping..."); }
}
0 references
class TestInheritance2
{
    0 references
    public static void Main(string[] args)
    {
        BabyDog d1 = new BabyDog();
        d1.eat();
        d1.bark();
        d1.weep();
    }
}
```



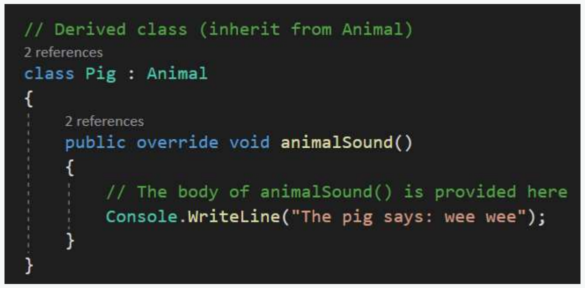
```
class Animal  // Base class (parent)
{
    0 references
    public void animalSound()
    {
        Console.WriteLine("The animal makes a sound");
    }
}

0 references
class Pig : Animal  // Derived class (child)
{
    0 references
    public void animalSound()
    {
        Console.WriteLine("The cat says: wee wee");
    }
}

0 references
class Dog : Animal  // Derived class (child)
{
    0 references
    public void animalSound()
    {
        Console.WriteLine("The dog says: bow wow");
    }
}
```

C# Abstraction

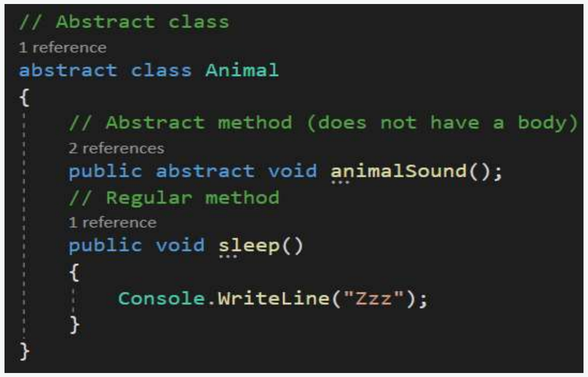
Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces.



Abstraction	Encapsulation
It is used to hide unwanted data and shows only the required properties and methods.	It binds data members and member functions into a single unit to prevent outsiders from accessing it directly.



```
// Derived class (inherit from Animal)
2 references
class Pig : Animal
{
    2 references
    public override void animalSound()
    {
        // The body of animalSound() is provided here
        Console.WriteLine("The pig says: wee wee");
    }
}
```

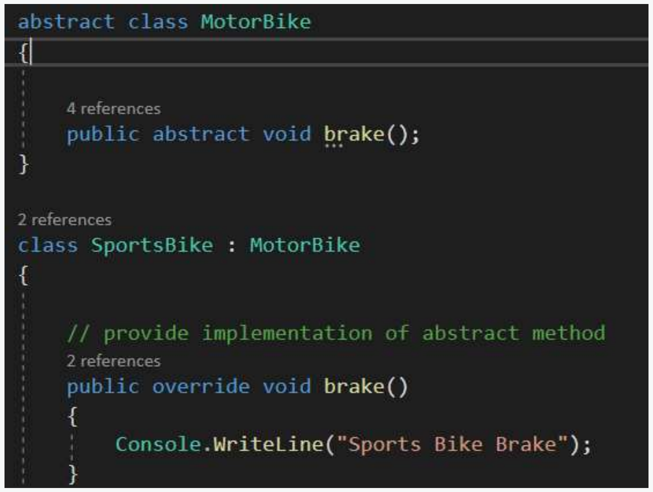


```
// Abstract class
1 reference
abstract class Animal
{
    // Abstract method (does not have a body)
    2 references
    public abstract void animalSound();
    // Regular method
    1 reference
    public void sleep()
    {
        Console.WriteLine("Zzz");
    }
}
```

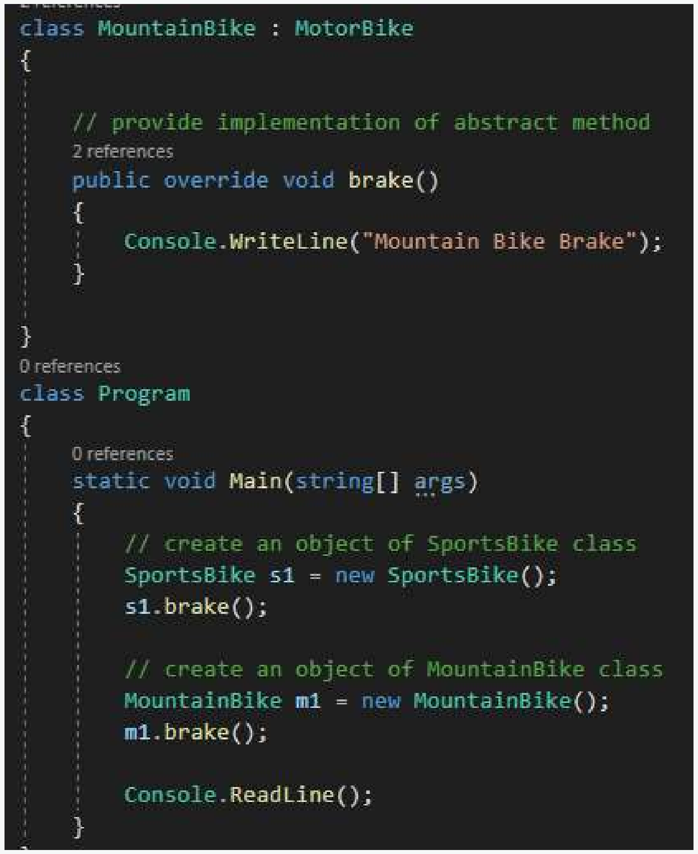
- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

C# Abstraction



```
abstract class MotorBike
{
    4 references
    public abstract void brake();
}

2 references
class SportsBike : MotorBike
{

    // provide implementation of abstract method
    2 references
    public override void brake()
    {
        Console.WriteLine("Sports Bike Brake");
    }
```



```
class MountainBike : MotorBike
{

    // provide implementation of abstract method
    2 references
    public override void brake()
    {
        Console.WriteLine("Mountain Bike Brake");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        // create an object of SportsBike class
        SportsBike s1 = new SportsBike();
        s1.brake();

        // create an object of MountainBike class
        MountainBike m1 = new MountainBike();
        m1.brake();

        Console.ReadLine();
    }
```

In the example, we have created an abstract class MotorBike.

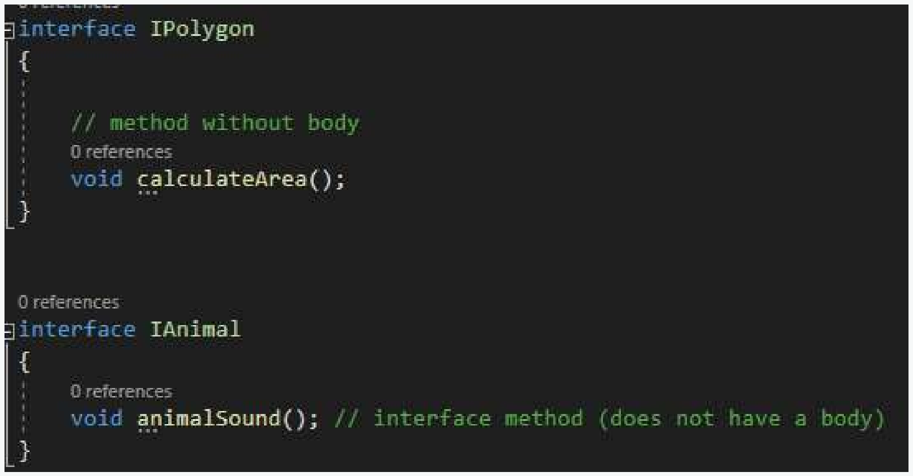
It has an abstract method brake(). As brake() is an abstract method the implementation of brake() in MotorBike is kept hidden.

Every motorbike has a different implementation of the brake.

This is why SportsBike makes its own implementation of brake() and MountainBike makes its own implementation of brake().

C# Interface

An interface is similar to abstract class. However, unlike abstract classes, all methods of an interface are fully abstract (method without body). We use the interface keyword to create an interface. For example,



```
interface IPolygon
{

    // method without body
    0 references
    void calculateArea();
}


0 references
interface IAnimal
{
    0 references
    void animalSound(); // interface method (does not have a body)
}
```

- IPolygon & IAnimal is the name of the interface.
- By convention, interface starts with I so that we can identify it just by seeing its name.
- We cannot use access modifiers inside an interface.
- All members of an interface are public by default.
- An interface doesn't allow fields.

Motive?

1) To achieve security - hide certain details and only show the important details of an object (interface).
2) C# does not support "multiple inheritance" (a class can only inherit from one base class). However, it can be achieved with interfaces, because the class can implement multiple interfaces.

C# Interface

```
using System;

namespace ooplab7
{
  interface IPolygon
  {
    // method without body
    void calculateArea();

  }
  // implements interface
  class Rectangle : IPolygon
  {

    // implementation of IPolygon interface
    public void calculateArea()
    {

      int l = 30;
      int b = 90;
      int area = l * b;
      Console.WriteLine("Area of Rectangle: " + area);
    }
  }

  class Square : IPolygon
  {

    // implementation of IPolygon interface
    public void calculateArea()
    {

      int l = 30;
      int area = l * l;
      Console.WriteLine("Area of Square: " + area);
    }
  }

  class Program
  {
    static void Main(string[] args)
    {

      Rectangle r1 = new Rectangle();
      r1.calculateArea();

      Square s1 = new Square();
      s1.calculateArea();
    }
  }
}
```

In the program, we have created an interface named IPolygon.

It has an abstract method calculateArea(). We have two classes Square and Rectangle that implement the IPolygon interface.

The rule for calculating the area is different for each polygon. Hence, calculateArea() is included without implementation.

Any class that implements IPolygon must provide an implementation of calculateArea().

Hence, implementation of the method in class Rectangle is independent of the method in class Square.

Let the debugging begins

Exercise: create class animal and cat (or any other animal you choose). Utilize abstraction and interface properties.