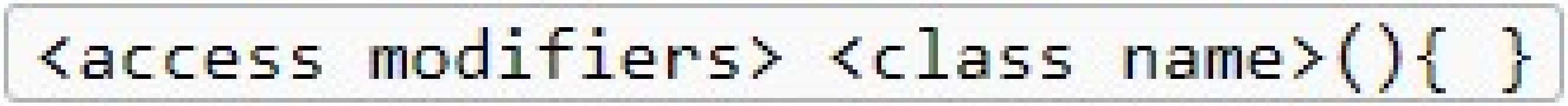
IT FACULTY

OOP WITH C#

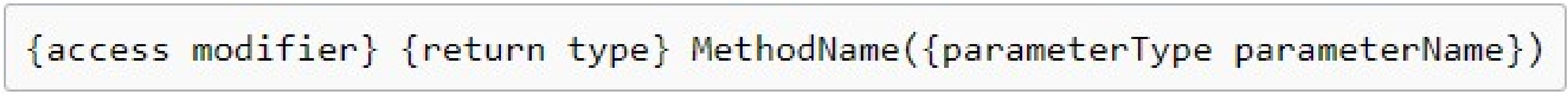
CLASS

- ***Class:***
  - Access Modifiers;
  - Field;
  - Constructor:



```
<access modifiers> <class name>(){ }
```

  - Method:

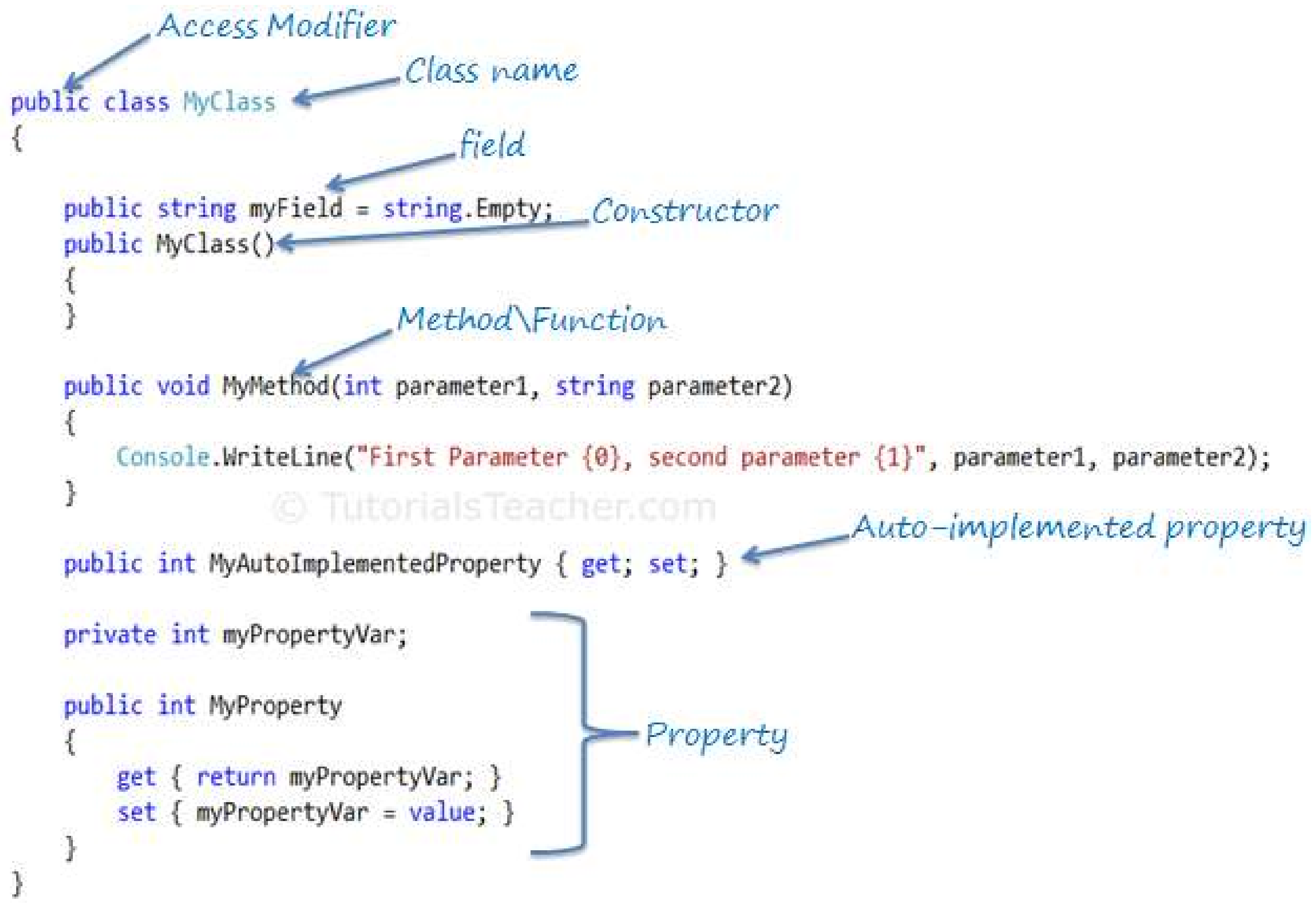


```
{access modifier} {return type} MethodName({parameterType parameterName})
```

  - Getter/Setter;
  - Property:
    - Auto-implemented Property.

CLASS

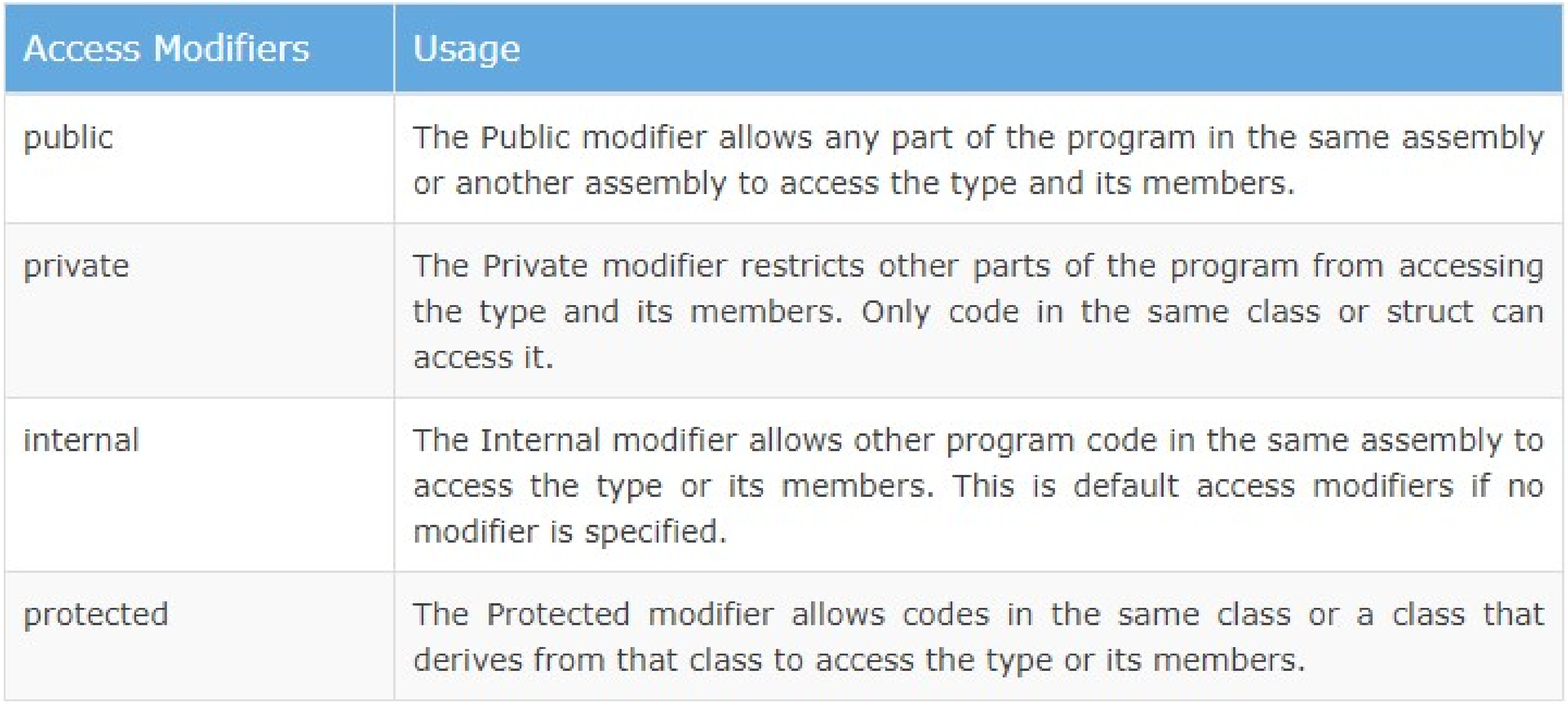
- ***Class:***



```
public class MyClass
{

    public string myField = string.Empty;
    public MyClass()
    {
    }

    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}", parameter1, parameter2);
    }

    public int MyAutoImplementedProperty { get; set; }

    private int myPropertyVar;

    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}
```

ACCESS MODIFIER

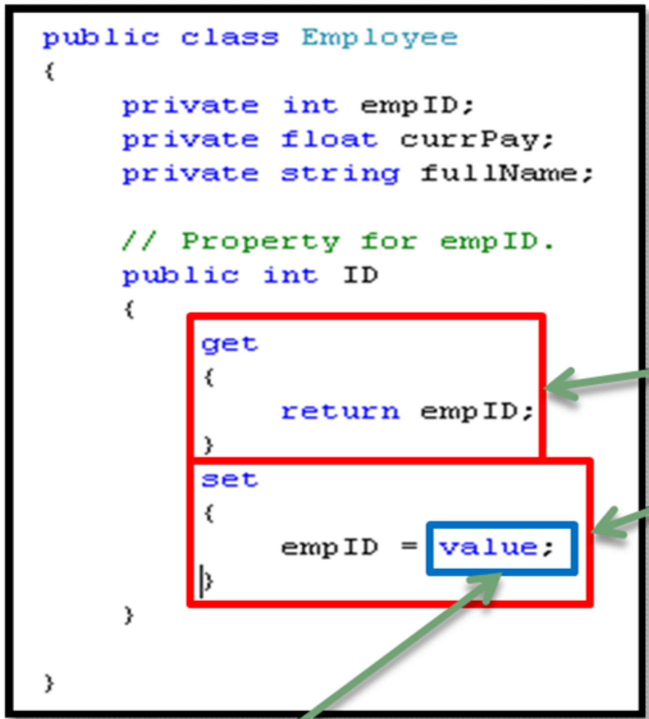
- Access modifiers are applied on the declaration of the class, method, properties, fields and other members.
- Define the accessibility of the class and its members.



Access Modifiers	Usage
public	The Public modifier allows any part of the program in the same assembly or another assembly to access the type and its members.
private	The Private modifier restricts other parts of the program from accessing the type and its members. Only code in the same class or struct can access it.
internal	The Internal modifier allows other program code in the same assembly to access the type or its members. This is default access modifiers if no modifier is specified.
protected	The Protected modifier allows codes in the same class or a class that derives from that class to access the type or its members.

IT FACULTY

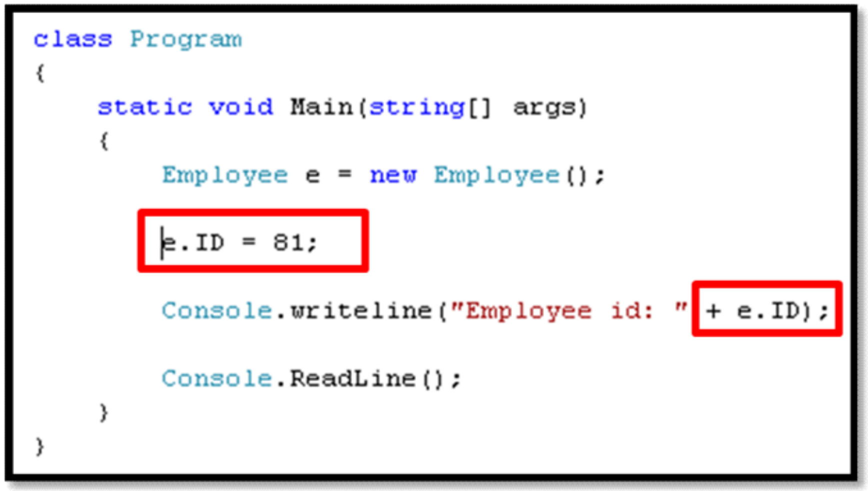
PROPERTY & ENCAPSULATION



```
public class Employee
{
    private int empID;
    private float currPay;
    private string fullName;

    // Property for empID.
    public int ID
    {
        get
        {
            return empID;
        }
        set
        {
            empID = value;
        }
    }

}
```

Using Properties

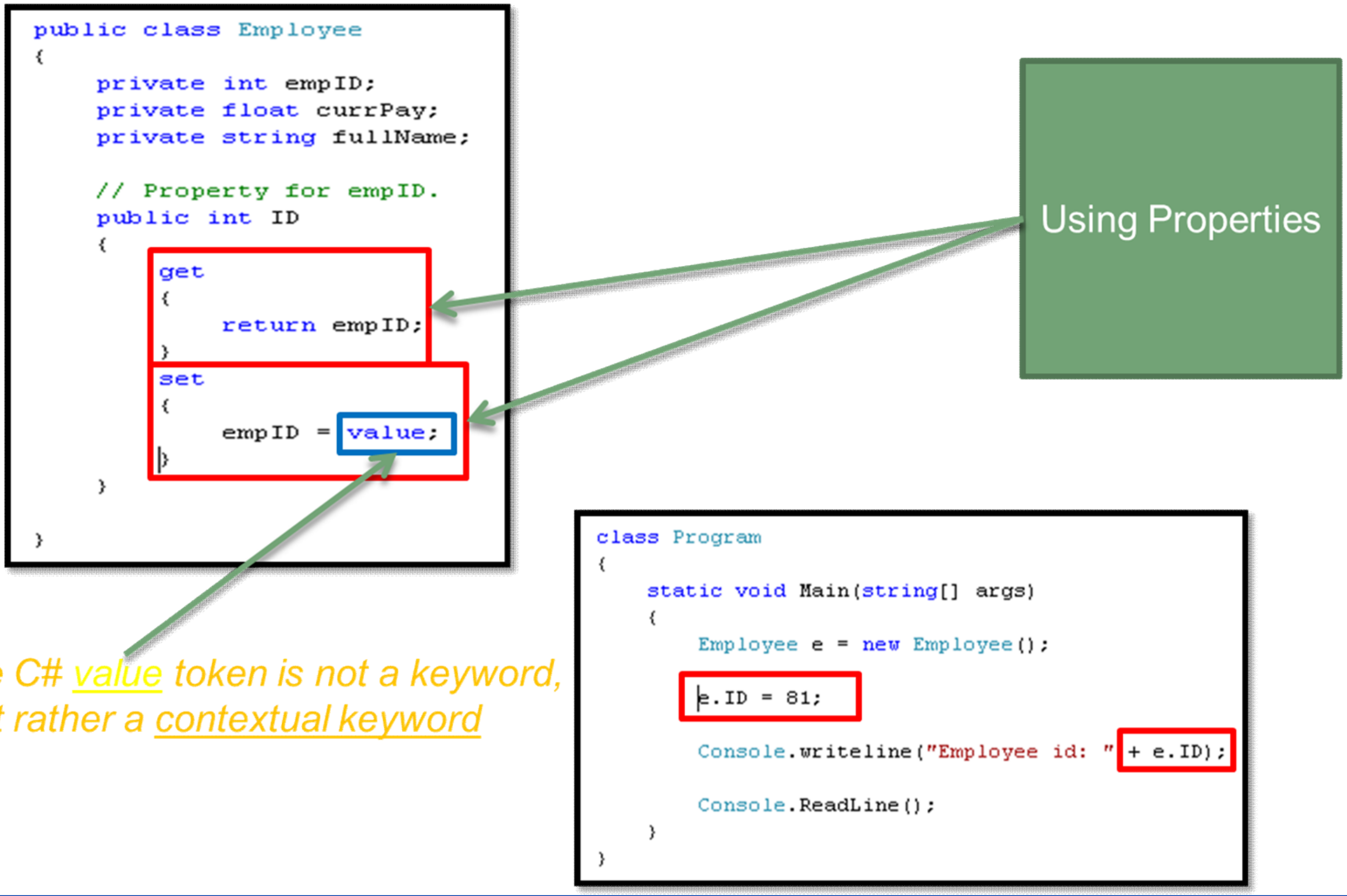
the C# value token is not a keyword, but rather a contextual keyword



```
class Program
{
    static void Main(string[] args)
    {
        Employee e = new Employee();

        e.ID = 81;

        Console.writeline("Employee id: " + e.ID);

        Console.ReadLine();
    }
}
```

PROPERTY & ENCAPSULATION

- A property can be defined using getters and setters:



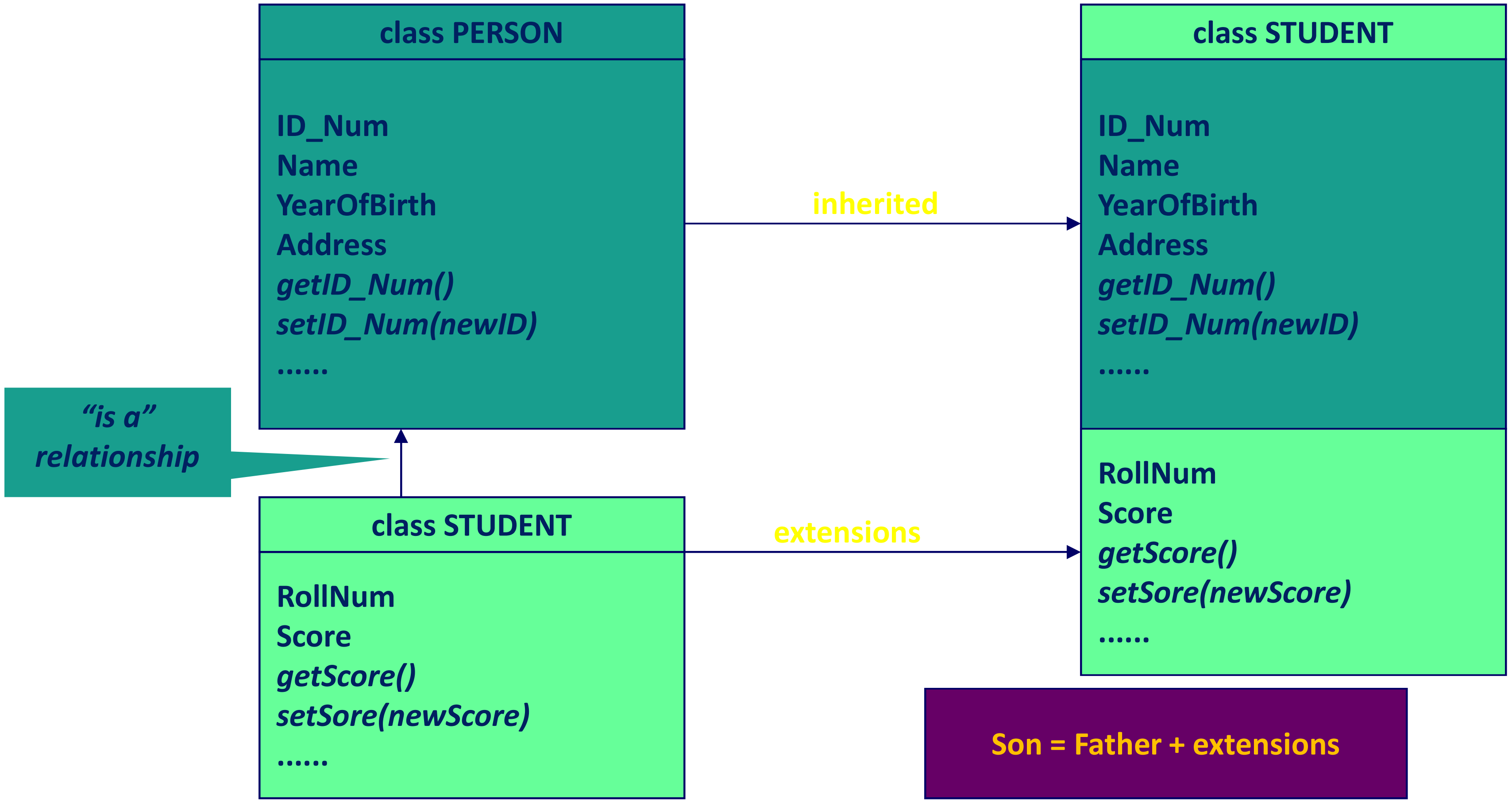
```
public class Employee
{
    private int empID;
    private float currPay;
    private string fullName;

    // Property for empID.
    public int ID
    {
        get
        {
            return empID;
        }
        set
        {
            empID = value;
        }
    }
}
```

the C# value token is not a keyword, but rather a contextual keyword

```
class Program
{
    static void Main(string[] args)
    {
        Employee e = new Employee();

        e.ID = 81;

        Console.writeline("Employee id: " + e.ID);

        Console.ReadLine();
    }
}
```

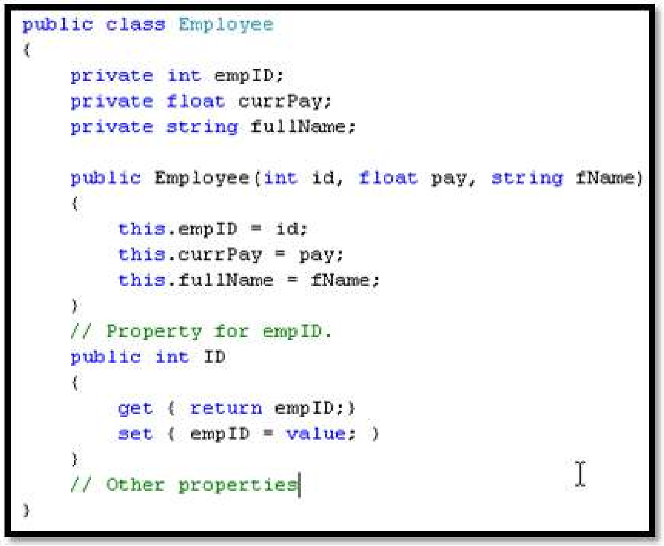
INHERITANCE

- Ability allows a class having members of an existed class → Re-used code.

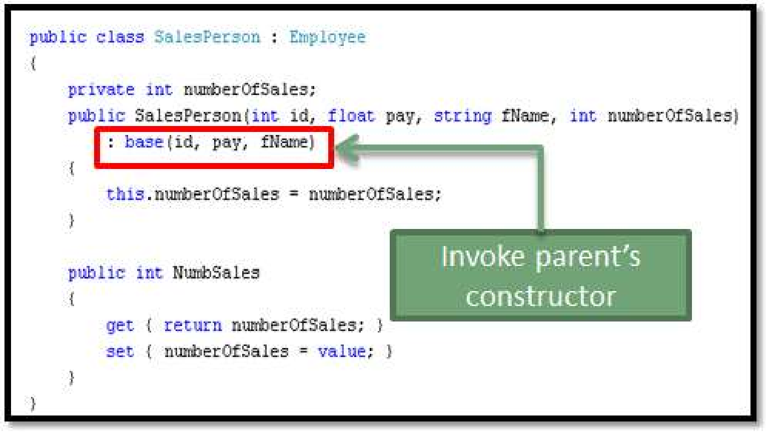


INHERITANCE

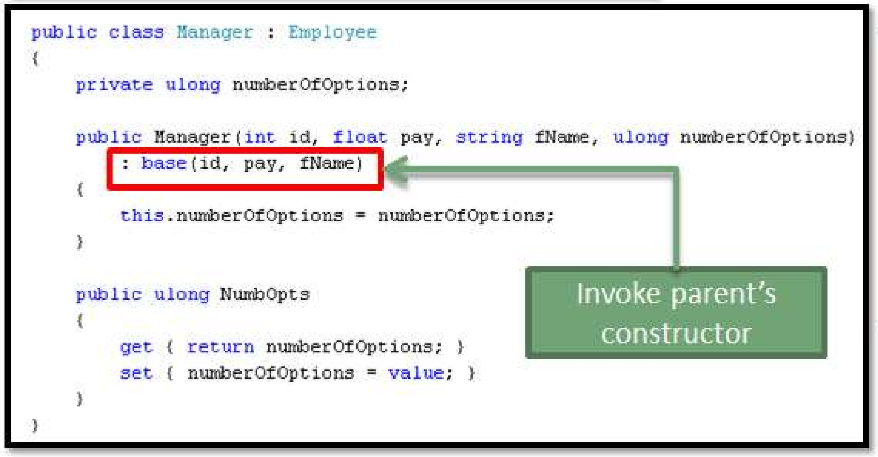
- C# and .NET support single inheritance only.



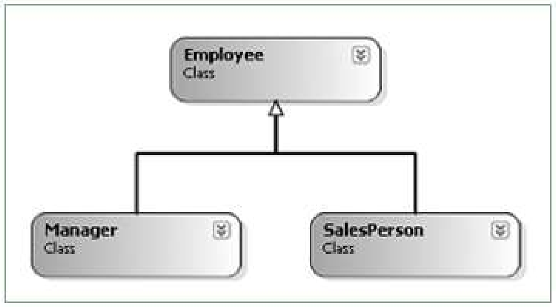
```
public class Employee
{
    private int empID;
    private float currPay;
    private string fullName;

    public Employee(int id, float pay, string fName)
    {
        this.empID = id;
        this.currPay = pay;
        this.fullName = fName;
    }
    // Property for empID.
    public int ID
    {
        get { return empID;}
        set { empID = value; }
    }
    // Other properties
}
```



```
public class SalesPerson : Employee
{
    private int numberOfSales;
    public SalesPerson(int id, float pay, string fName, int numberOfSales)
        : base(id, pay, fName)
    {
        this.numberOfSales = numberOfSales;
    }

    public int NumbSales
    {
        get { return numberOfSales; }
        set { numberOfSales = value; }
    }
}
```

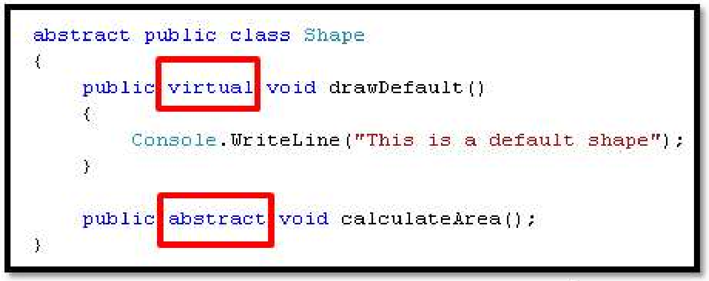


```
public class Manager : Employee
{
    private ulong numberOfOptions;

    public Manager(int id, float pay, string fName, ulong numberOfOptions)
        : base(id, pay, fName)
    {
        this.numberOfOptions = numberOfOptions;
    }

    public ulong NumbOpts
    {
        get { return numberOfOptions; }
        set { numberOfOptions = value; }
    }
}
```



POLYMORPHISM

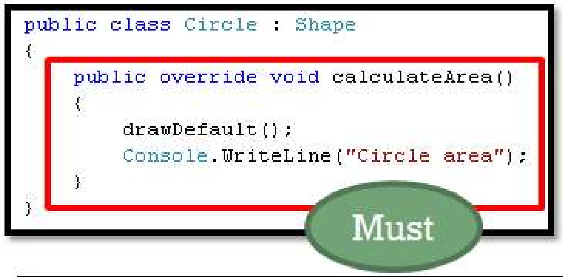
- Overload & Override (keyword **new** ???)



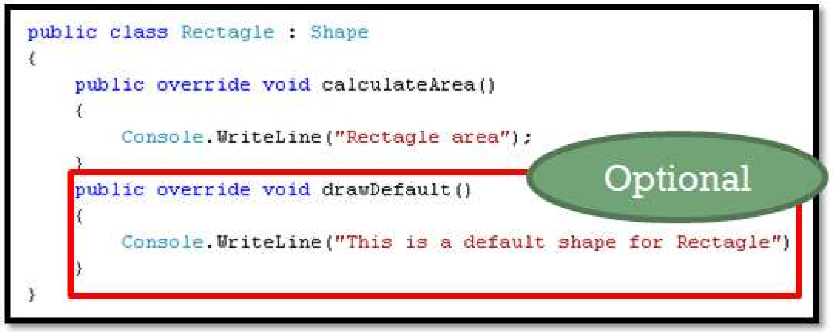
```
abstract public class Shape
{
    public virtual void drawDefault()
    {
        Console.WriteLine("This is a default shape");
    }

    public abstract void calculateArea();
}
```

virtual : provide a default implementation. Can be overridden if necessary

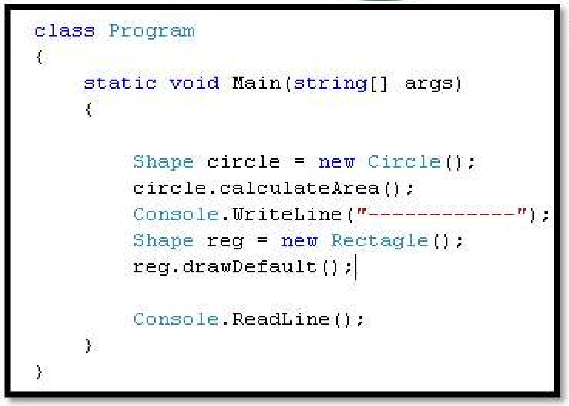
abstract: sub-classes MUST override



```
public class Circle : Shape
{
    public override void calculateArea()
    {
        drawDefault();
        Console.WriteLine("Circle area");
    }
}
```

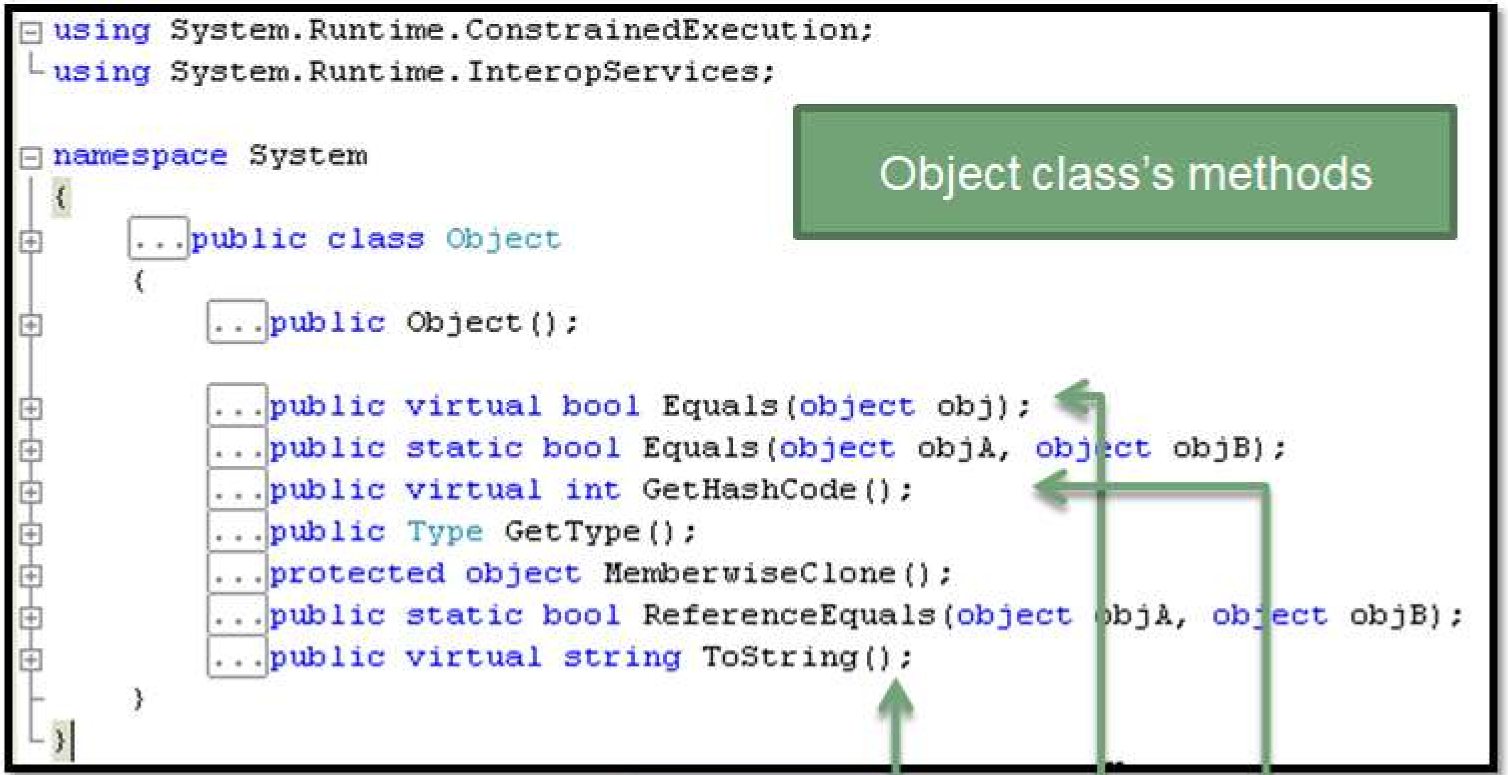


```
public class Rectagle : Shape
{
    public override void calculateArea()
    {
        Console.WriteLine("Rectagle area");
    }
    public override void drawDefault()
    {
        Console.WriteLine("This is a default shape for Rectagle")
    }
}
```



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

        Shape circle = new Circle();
        circle.calculateArea();
        Console.WriteLine("------------");
        Shape reg = new Rectagle();
        reg.drawDefault();

        Console.ReadLine();
    }
}
```

CLASS OBJECT

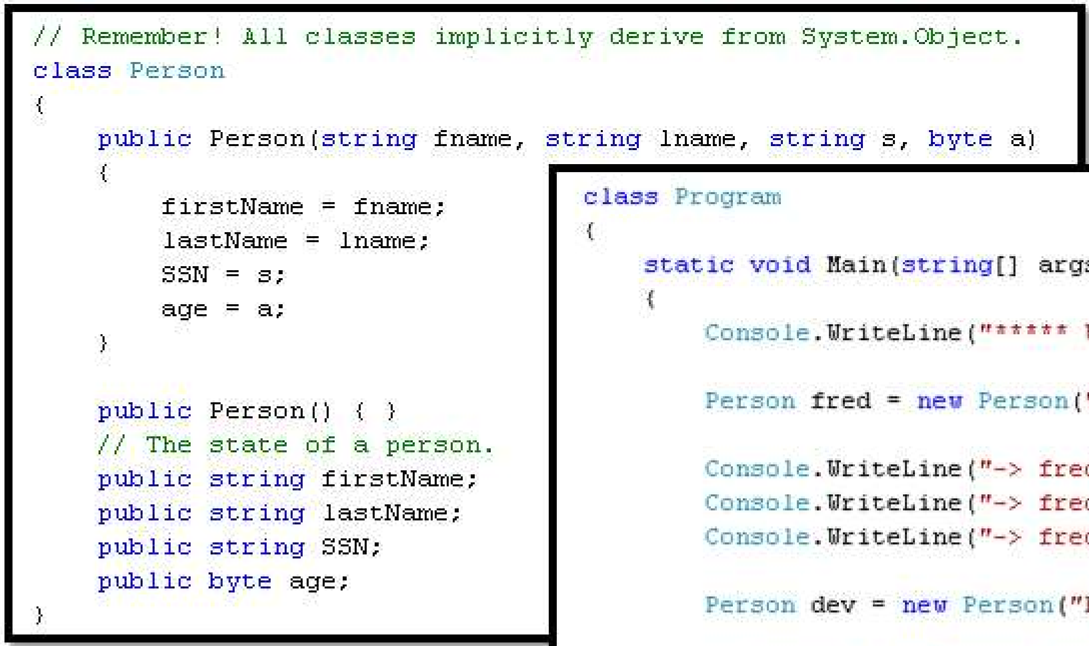


```
using System.Runtime.ConstrainedExecution;
using System.Runtime.InteropServices;

namespace System
{
    ...public class Object
    {
        ...public Object();

        ...public virtual bool Equals(object obj);
        ...public static bool Equals(object objA, object objB);
        ...public virtual int GetHashCode();
        ...public Type GetType();
        ...protected object MemberwiseClone();
        ...public static bool ReferenceEquals(object objA, object objB);
        ...public virtual string ToString();
    }
}
```

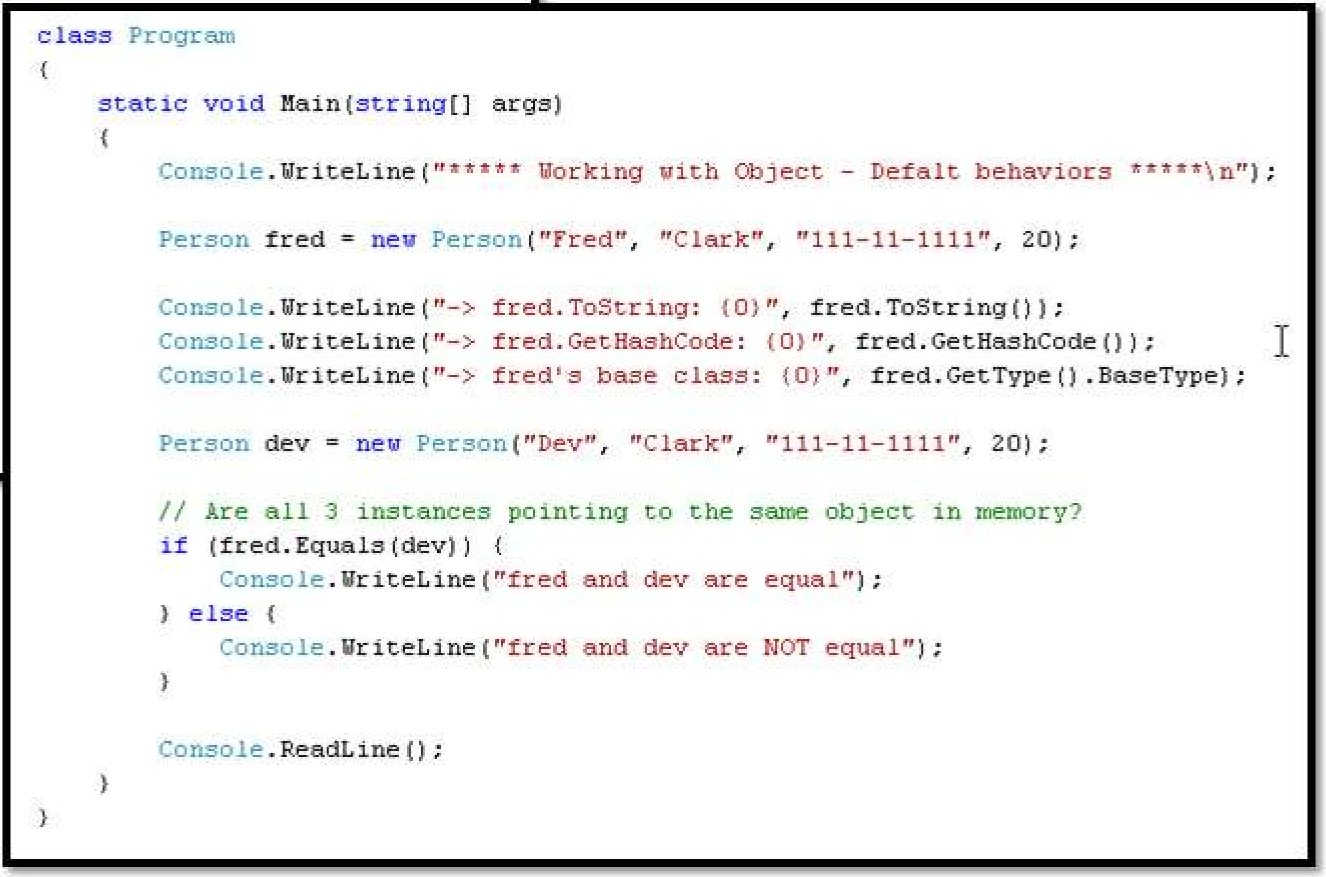
Object class's methods

We usually override these methods

CLASS OBJECT: OVERRIDE

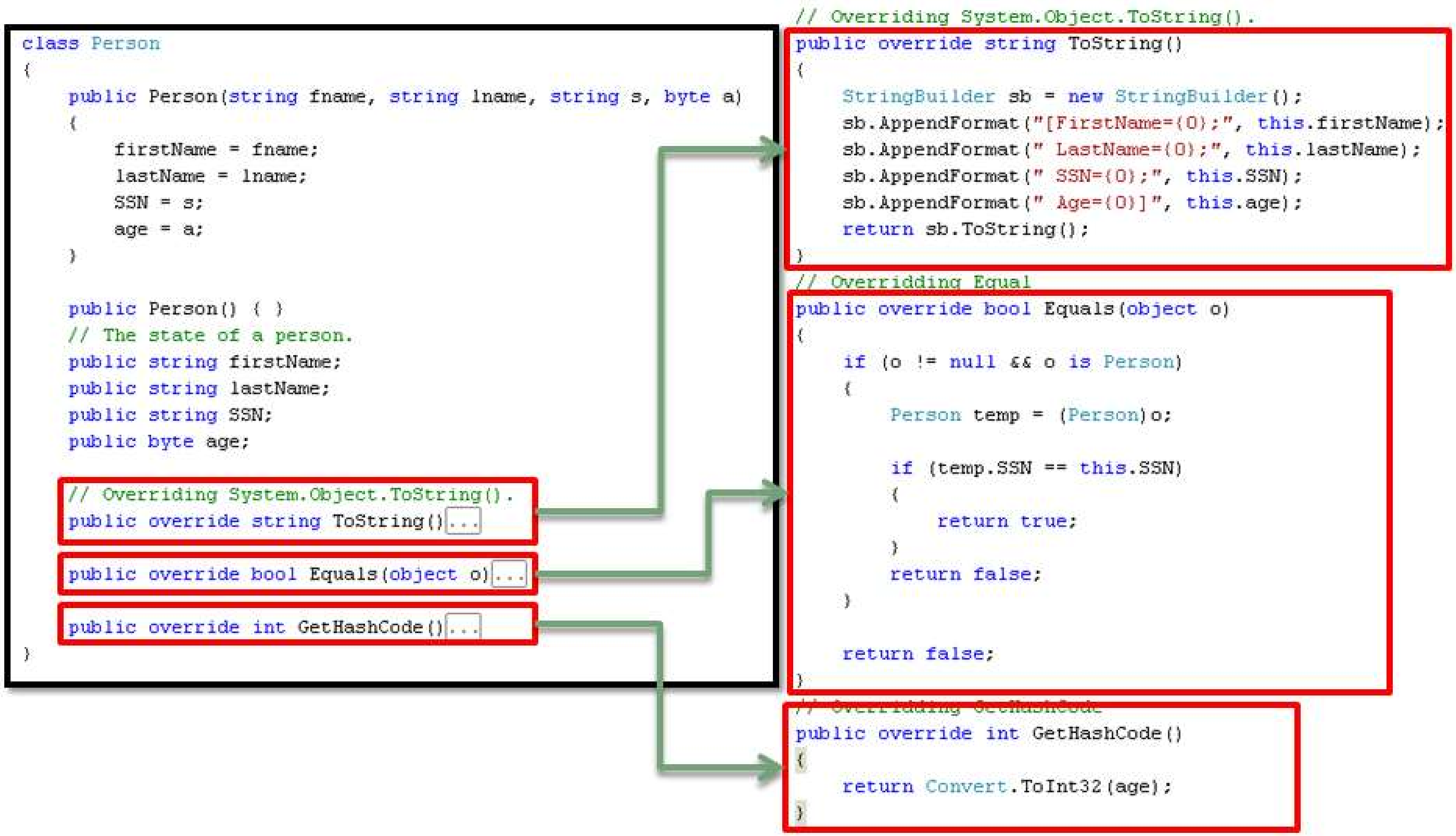


```
// Remember! All classes implicitly derive from System.Object.
class Person
{
    public Person(string fname, string lname, string s, byte a)
    {
        firstName = fname;
        lastName = lname;
        SSN = s;
        age = a;
    }

    public Person() { }
    // The state of a person.
    public string firstName;
    public string lastName;
    public string SSN;
    public byte age;
}
```



```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("***** Working with Object - Defalt behaviors *****\n");

        Person fred = new Person("Fred", "Clark", "111-11-1111", 20);

        Console.WriteLine("-> fred.ToString: {0}", fred.ToString());
        Console.WriteLine("-> fred.GetHashCode: {0}", fred.GetHashCode());
        Console.WriteLine("-> fred's base class: {0}", fred.GetType().BaseType);

        Person dev = new Person("Dev", "Clark", "111-11-1111", 20);

        // Are all 3 instances pointing to the same object in memory?
        if (fred.Equals(dev)) {
            Console.WriteLine("fred and dev are equal");
        } else {
            Console.WriteLine("fred and dev are NOT equal");
        }

        Console.ReadLine();
    }
}
```

CLASS OBJECT: OVERRIDE

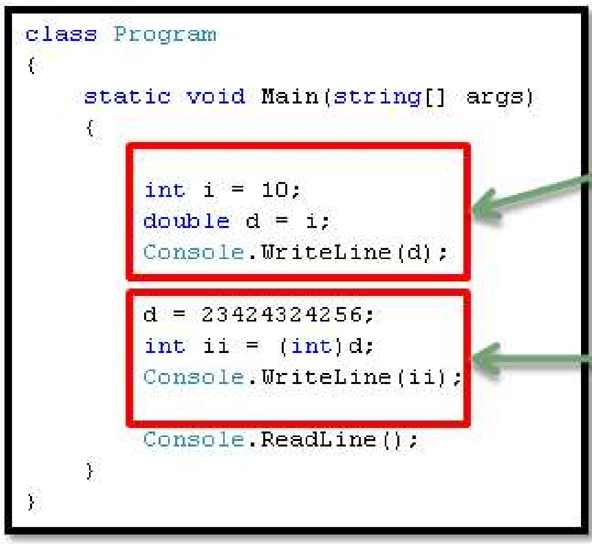


```
class Person
{
    public Person(string fname, string lname, string s, byte a)
    {
        firstName = fname;
        lastName = lname;
        SSN = s;
        age = a;
    }

    public Person() { }
    // The state of a person.
    public string firstName;
    public string lastName;
    public string SSN;
    public byte age;

    // Overriding System.Object.ToString().
    public override string ToString()...

    public override bool Equals(object o)...

    public override int GetHashCode()...
}
```

```
// Overriding System.Object.ToString().
public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    sb.AppendFormat("[FirstName={0};", this.firstName);
    sb.AppendFormat(" LastName={0};", this.lastName);
    sb.AppendFormat(" SSN={0};", this.SSN);
    sb.AppendFormat(" Age={0}]", this.age);
    return sb.ToString();
}
// Overridding Equal
public override bool Equals(object o)
{
    if (o != null && o is Person)
    {
        Person temp = (Person)o;

        if (temp.SSN == this.SSN)
        {
            return true;
        }
        return false;
    }

    return false;
}
// Overridding GetHashCode
public override int GetHashCode()
{
    return Convert.ToInt32(age);
}
```

CASTING

- Value type: conversion and casting

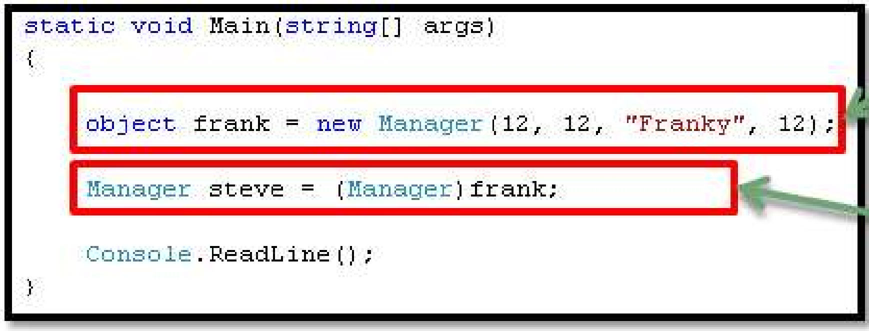


```
class Program
{
    static void Main(string[] args)
    {
        int i = 10;
        double d = i;
        Console.WriteLine(d);

        d = 23424324256;
        int ii = (int)d;
        Console.WriteLine(ii);

        Console.ReadLine();
    }
}
```

Convert smaller type to bigger type: OK (also called implicit cast)

Convert bigger type to smaller type: NOT OK => need an explicit cast => *may cause loss of data*

- Reference type: conversion and casting



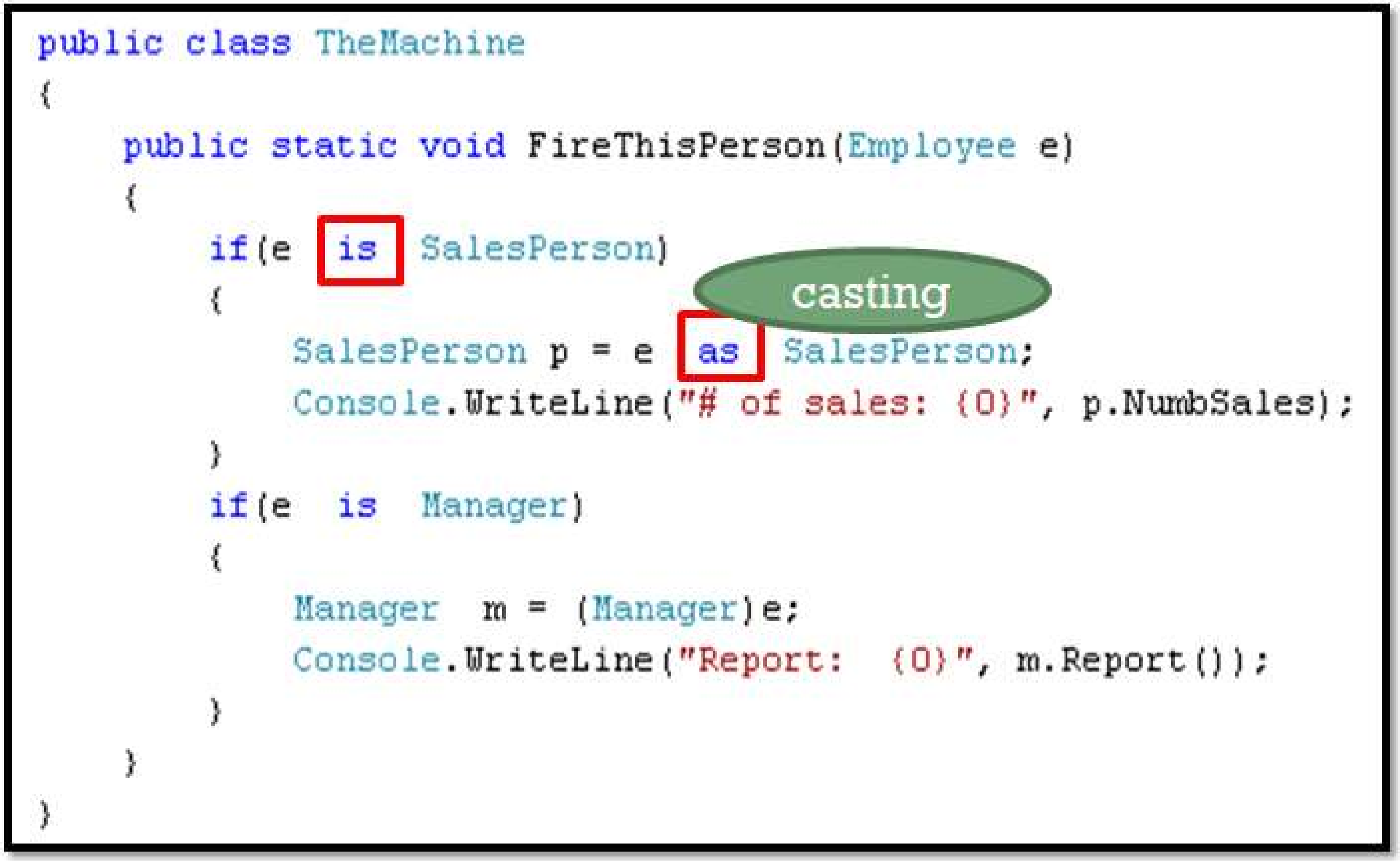
```
static void Main(string[] args)
{
    object frank = new Manager(12, 12, "Franky", 12);

    Manager steve = (Manager)frank;

    Console.ReadLine();
}
```

Convert sub-class to supper class: OK (also called implicit cast)

Convert supper class to sub-class: NOT OK => need an explicit cast => *may cause run time error*

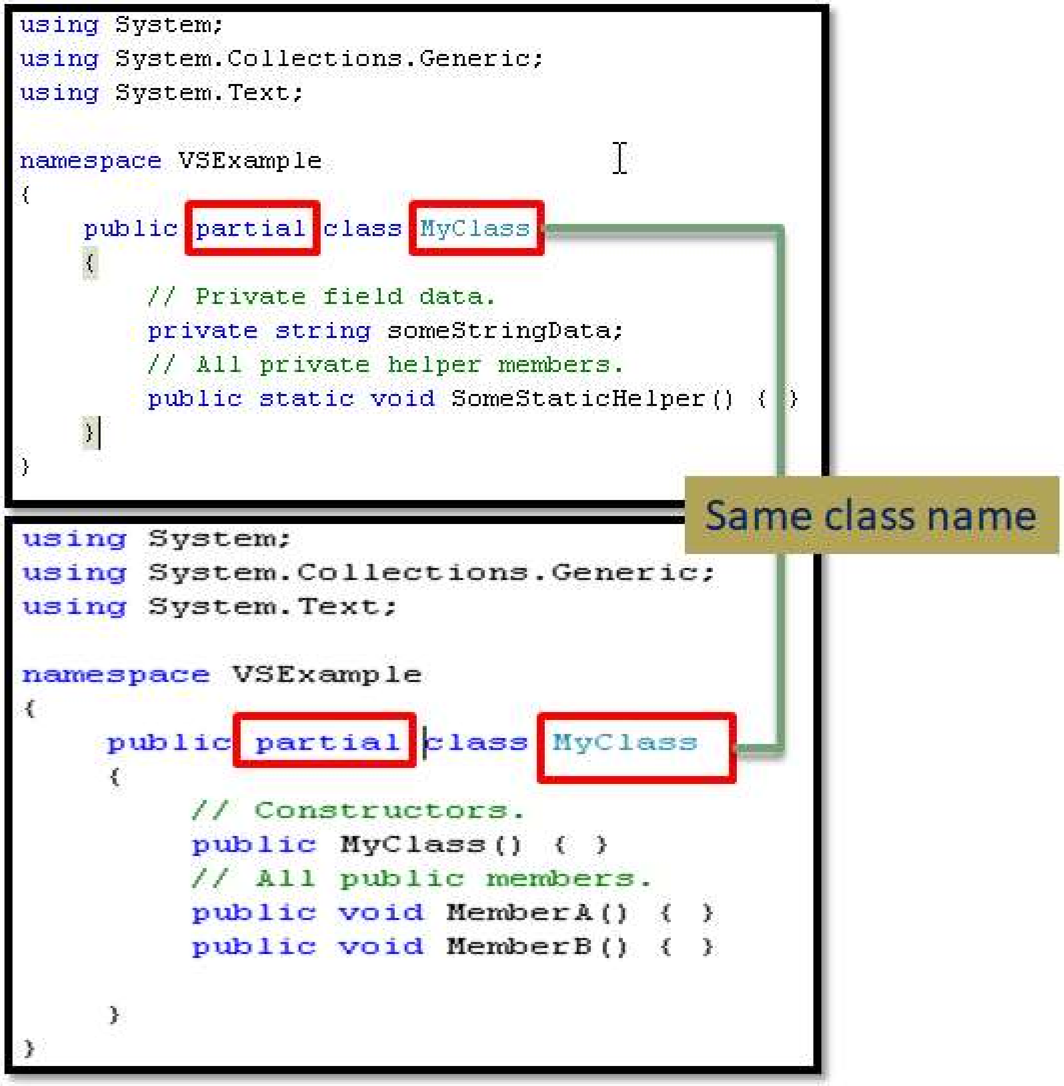
DETERMINING THE "TYPE OF"

- *is*
- *as*



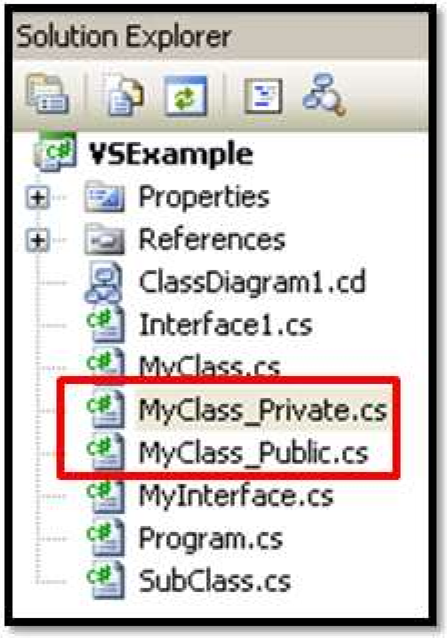
```
public class TheMachine
{
    public static void FireThisPerson(Employee e)
    {
        if(e is SalesPerson)
        {
            SalesPerson p = e as SalesPerson;
            Console.WriteLine("# of sales: {0}", p.NumbSales);
        }
        if(e  is  Manager)
        {
            Manager  m = (Manager)e;
            Console.WriteLine("Report:  {0}", m.Report());
        }
    }
}
```

PARTIAL TYPES



```
using System;
using System.Collections.Generic;
using System.Text;

namespace VSExample
{
    public partial class MyClass
    {
        // Private field data.
        private string someStringData;
        // All private helper members.
        public static void SomeStaticHelper() { }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace VSExample
{
    public partial class MyClass
    {
        // Constructors.
        public MyClass() { }
        // All public members.
        public void MemberA() { }
        public void MemberB() { }

    }
}
```



Thank You !

IT FACULTY