Introduction to OOP

Classes and Objects

C#

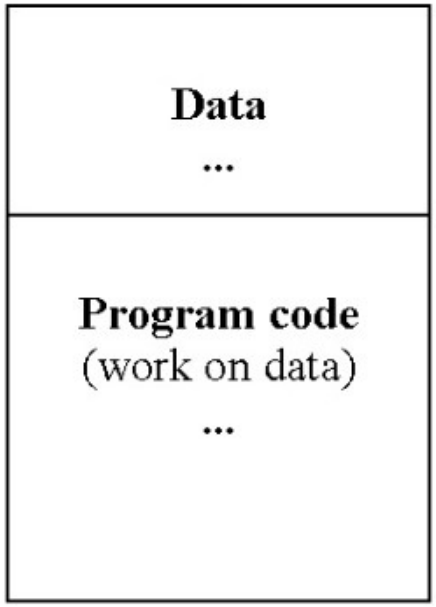
```
class SampleClass
{
}
```

Structures

C#

```
struct SampleStruct
{
}
```

Class Members



Fields member

C#

```
class SampleClass
{
    public string sampleField;
}
```

Properties

C#

```
class SampleClass
{
    private int _sample;
    public int Sample
    {
        // Return the value stored in a field.
        get { return _sample; }
        // Store the value in the field.
        set { _sample = value; }
    }
}
```

C#

```
class SampleClass
{
    public int SampleProperty { get; set; }
}
```

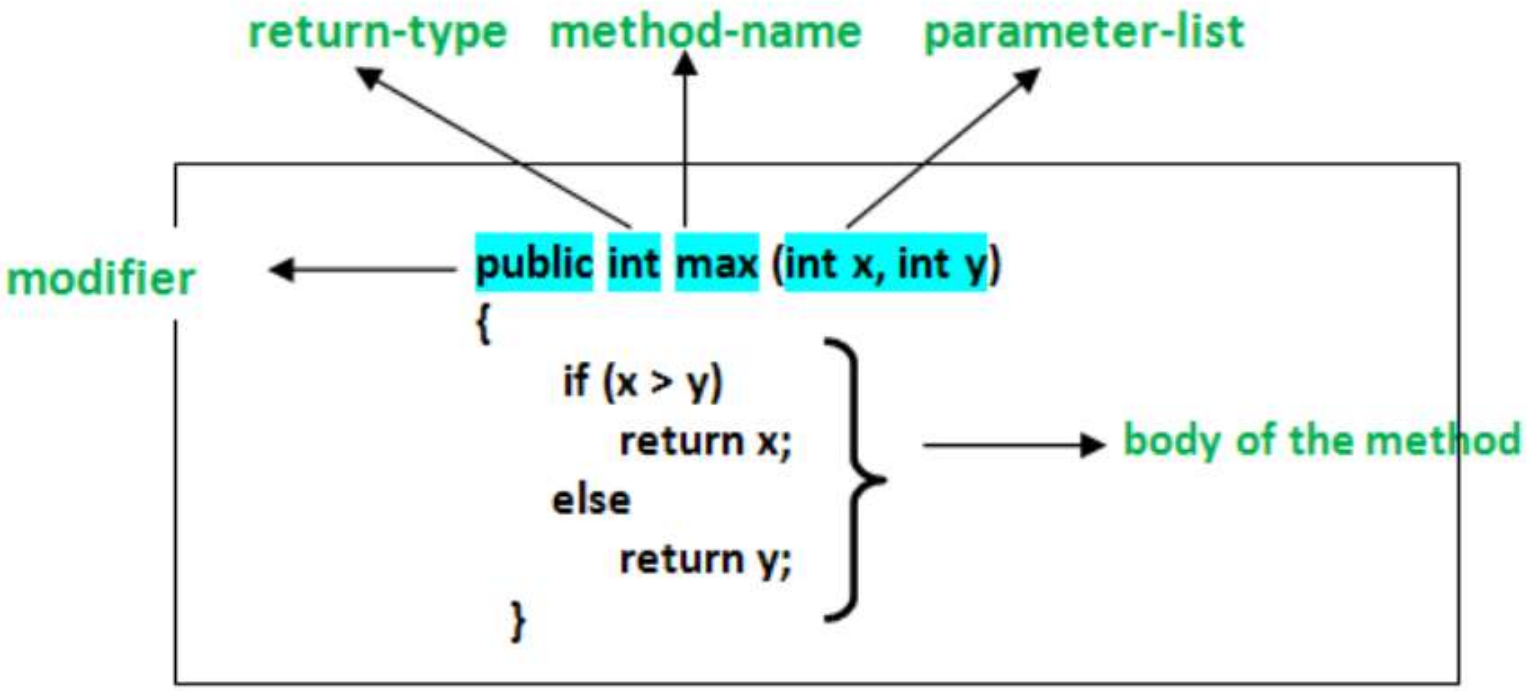
Methods

C#

```
class SampleClass
{
    public int sampleMethod(string sampleParam)
    {
        // Insert code here
    }
}
```

Method Signature

Name Parameters typese



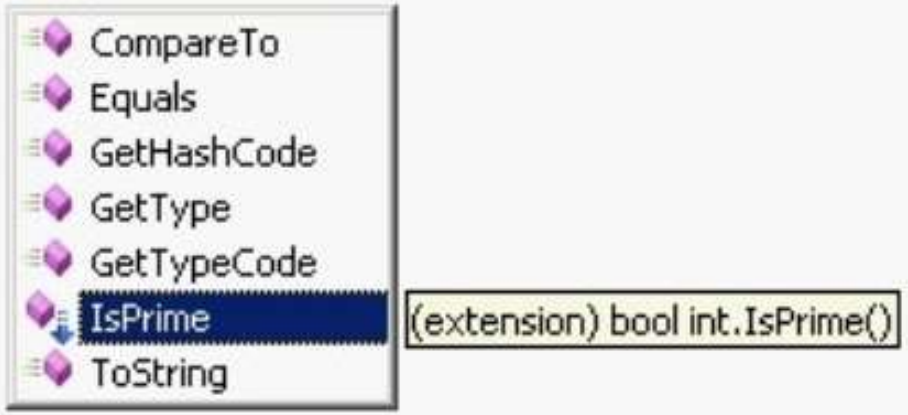
Method overloading

C#

```
public int sampleMethod(string sampleParam) {};
public int sampleMethod(int sampleParam) {}
```

Extension Method

```
class Program
{
    static void Main(string[] args)
    {
        int anyNumber = 123456;
        anyNumber.
    }
}
```



Method Constructors

```
public class SampleClass
{
    public SampleClass()
    {
        // Add code here
    }
}
```

Nested Classes

```
class Container
{
    class Nested
    {
        // Add code here.
    }
}
```

Create object of nested class

```
Container.Nested nestedInstance = new Container.Nested()
```

Access Modifiers

Modifier	Definition
public	The type or member can be accessed by any other code in the same assembly or another assembly that references it.
private	The type or member can only be accessed by code in the same class.
protected	The type or member can only be accessed by code in the same class or in a derived class.
internal	The type or member can be accessed by any code in the same assembly, but not from another assembly.
protected internal	The type or member can be accessed by any code in the same assembly, or by any derived class in another assembly.
private protected	The type or member can be accessed by code in the same class or in a derived class within the base class assembly.

Instantiating Classes

```
SampleClass sampleObject = new SampleClass();
```

Using class members

```
// Set a property value.
sampleObject.sampleProperty = "Sample String";
// Call a method.
sampleObject.sampleMethod();
```

Object initializers

```
// Set a property value.
SampleClass sampleObject = new SampleClass
    { FirstProperty = "A", SecondProperty = "B" };
```

Static Members

```
static class SampleClass
{
    public static string SampleString = "Sample String";
}
```

Accessing Static Member

```
Console.WriteLine(SampleClass.SampleString);
```

Anonymous Types

Amount

Message

```
var v = new { Amount = 108, Message = "Hello" };

// Rest the mouse pointer over v.Amount and v.Message in the following
// statement to verify that their inferred types are int and string.
Console.WriteLine(v.Amount + v.Message);
```

Inheritance

base class

derived class

```
class DerivedClass:BaseClass {}
```

Sealed Class

```
public sealed class A { }
```

Abstract Class

```
public abstract class B { }
```

Overriding Members

```
16      class foo
17      {
18          public int i = 1;
19          public virtual void changeI()
20          {
21              i++;
22          }
23      }
24      class bar : foo
25      {
26          override public void changeI()
27          {
28              i--;
29          }
30      }
31  }
```

C# Modifier	Definition
virtual	Allows a class member to be overridden in a derived class.
override	Overrides a virtual (overridable) member defined in the base class.
abstract	Requires that a class member to be overridden in the derived class.
new Modifier	Hides a member inherited from a base class

Interfaces

Feature	Interface	Abstract class
Multiple inheritance	A class may inherit several interfaces.	A class may inherit only one abstract class.
Default implementation	An interface cannot provide any code, just the signature.	An abstract class can provide complete, default code and/or just the details that have to be overridden.
Access Modfiers	An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties

Generics

```
public class SampleGeneric<T>
{
    public T Field;
}
```

```
SampleGeneric<string> sampleObject = new SampleGeneric<string>();
sampleObject.Field = "Sample string";
```

Delegates

```
public delegate void SampleDelegate(string str);
```

```
class SampleClass
{
    // Method that matches the SampleDelegate signature.
    public static void sampleMethod(string message)
    {
        // Add code here.
    }
    // Method that instantiates the delegate.
    void SampleDelegate()
    {
        SampleDelegate sd = sampleMethod;
        sd("Sample string");
    }
}
```