



Introduction to C# object-oriented programming



Object Oriented Programming

Why C# ?

- Native support for
 - Namespaces
 - Versioning
 - Attribute-driven development
- Power of C with ease of Microsoft Visual Basic®
- Minimal learning curve for everybody
- Much cleaner than C++
- More structured than Visual Basic
- More powerful than Java

C# – The Big Ideas

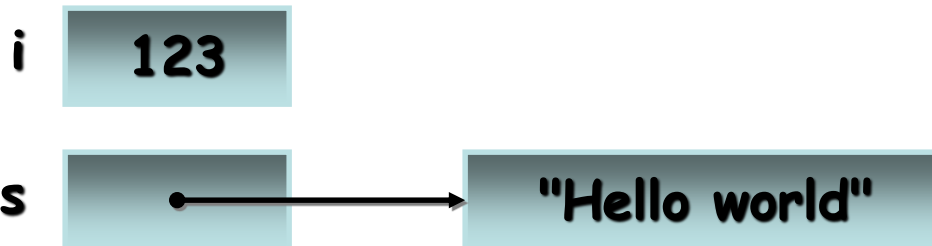
- A component oriented language
- The first “component oriented” language in the C/C++ family
 - In OOP a component is: A reusable program that can be combined with other components in the same system to form an application.
 - They can be deployed on different servers and communicate with each other
- Enables one-stop programming
 - No header files, IDL, etc.
 - Can be embedded in web pages

Value and Reference Types

Definition

- Value types
 - Directly contain data
 - Cannot be null
- Reference types
 - Contain references to objects
 - May be null

```
int i = 123;  
string s = "Hello world";
```



Value and Reference Types

Example

- Value types
 - Primitives
 - Enums
 - Structs
- Reference types
 - Classes
 - Interfaces
 - Arrays
 - Delegates

```
int i;
```

```
enum State { Off, On }
```

```
struct Point { int x, y; }
```

```
class Foo: Bar, IFoo {...}
```

```
interface IFoo: IBar {...}
```

```
string[] a = new string[10];
```

```
delegate void Empty();
```

Value and Reference Types

Classes

- Single inheritance
- Multiple interface implementation
- Class members
 - Constants, fields, methods, properties, indexers, events, operators, constructors, destructors
 - Static and instance members
 - Nested types
- Member access
 - Public, protected, internal, private

Value and Reference Types

Structs

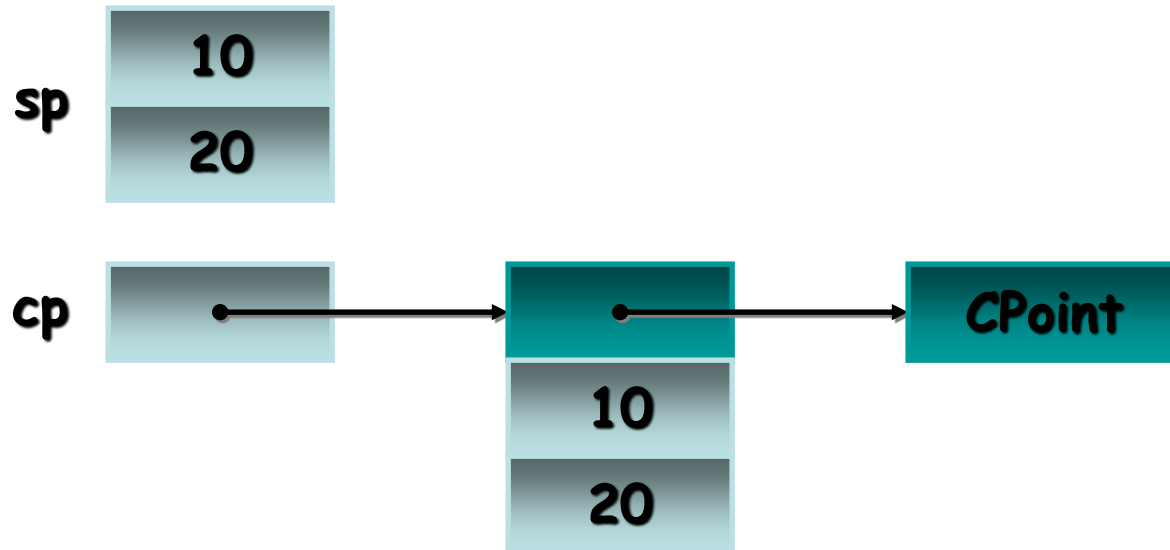
- Like classes, except
 - Stored in-line, not heap allocated
 - Assignment copies data, not reference
 - No inheritance
- Ideal for light weight objects
 - Complex, point, rectangle, color
 - int, float, double, etc., are all structs
- Benefits
 - No heap allocation, less GC pressure
 - More efficient use of memory

Value and Reference Types

Classes and Structs

```
class CPoint { int x, y; ... }  
struct SPoint { int x, y; ... }
```

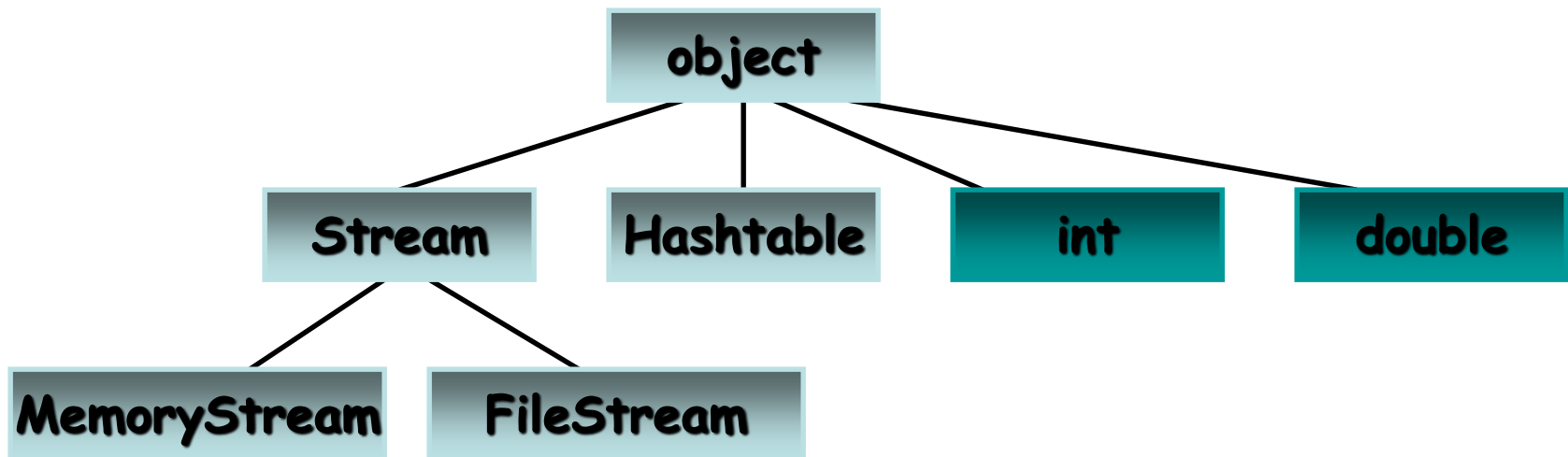
```
CPoint cp = new CPoint(10, 20);  
SPoint sp = new SPoint(10, 20);
```



Value and Reference Types

Unified Type System

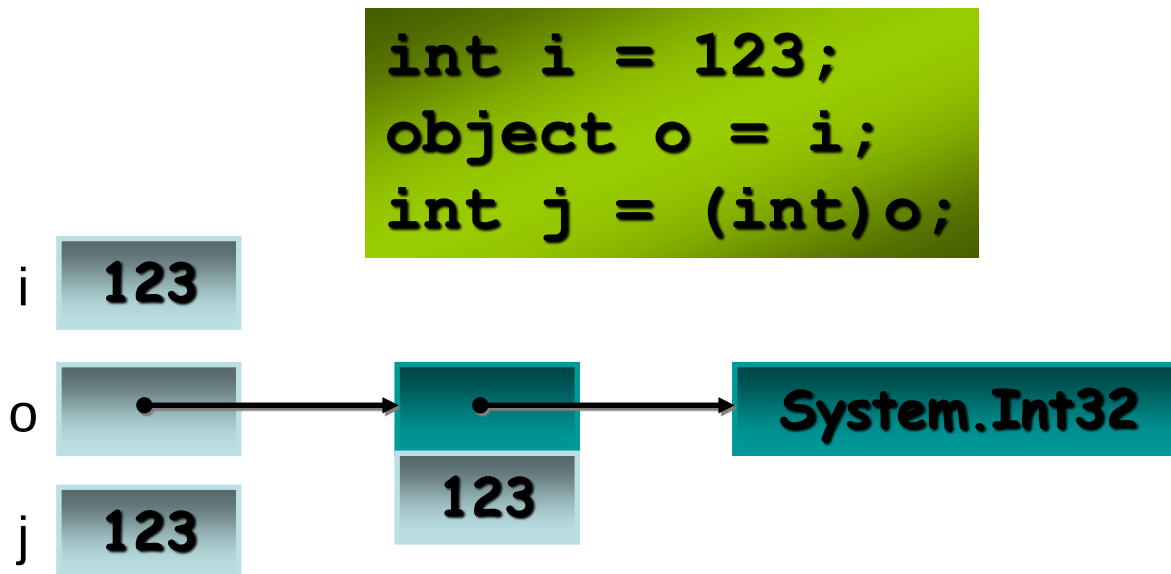
- Everything is an object
 - All types ultimately inherit from object
 - Any piece of data can be stored, transported, and manipulated with no extra work



Value and Reference Types

Boxing and Unboxing

- Boxing
 - Allocates box, copies value into it
- Unboxing
 - Checks type of box, copies value out



Language Features

Unified Type System

- Benefits
 - Eliminates “wrapper classes”
 - Collection classes work with all types
 - Replaces OLE Automation's Variant
- Lots of examples in .NET framework

```
string s = string.Format(
    "Your total was {0} on {1}", total, date);

Hashtable t = new Hashtable();
t.Add(0, "zero");
t.Add(1, "one");
t.Add(2, "two");
```

Classes

- Way of organizing data
- Can have:
 - member data
 - member methods

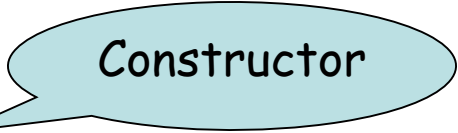
```
Class clsName
{
    modifier dataType varName;
    modifier returnType methodName (params)
    {
        statements;
        return returnVal;
    }
}
```

Example of the class

```
namespace Sample
{
    using System;

    public class HelloWorld
    {
        public HelloWorld()
        {
        }

        public static int Main(string[] args)
        {
            Console.WriteLine("Hello world!");
            return 0;
        }
    }
}
```



Constructor

Example of the class

```
using System;
namespace ConsoleTest
{
    public class Class1
    {
        public string FirstName = "Kay";
        public string LastName = "Connelly";

        public string GetWholeName()
        {
            return FirstName + " " + LastName;
        }

        static void Main(string[] args)
        {
            Class1 myClassInstance = new Class1();

            Console.WriteLine("Name: " + myClassInstance.GetWholeName());

            while(true) ;
        }
    }
}
```

Base Classes

Definition

- Synonymous with Superclass
- Derived Classes are created from Base classes.
- Base classes may also be derived classes
- Keywords
 - private
 - public
 - internal
 - protected
 - virtual

Derived Classes

Definition

- Synonymous with Subclass
- Single Inheritance
- Extend base class functionality
- Replace base class functionality
- Augment existing functionality
- Keywords
 - private
 - public
 - internal
 - protected
 - override
 - base
 - sealed

Base & Derived Classes

Syntax

```
public class Mammal
{
    public void Breathe() {...}
    public virtual void Walk() {...}
    public virtual void Eat() {...}
}

public class Dog: Mammal
{
    override public void Walk() {...}
    public string Bark() {...}
}
```

Abstract Classes

Definition

- A base class that can not be created
- Must be overridden
- Keywords
 - private
 - public
 - internal
 - protected
 - virtual
 - abstract (class and member level)

Abstract & Derived Classes

Example

```
public abstract class Mammal
{
    public void Breathe() {...}
    public virtual void Walk() {...}
    public virtual void Eat() {...}
    public abstract bool Swim()
}

public class Dog: Mammal
{
    override public void Walk() {...}
    public string Bark() {...}
    override public bool Swim() {return false;}
}
```

Interfaces

Definition

- Multiple inheritance
- Can contain methods, properties, indexers and events
- Provides polymorphic behavior
- Does not represent an 'is-a' relationship
- Represents a contract between classes

Interfaces

Syntax

```
interface IDataBound
{
    void Bind(IDataBinder binder);
}

class EditBox: Control, IDataBound
{
    void IDataBound.Bind(IDataBinder binder) {...}
}
```

Interfaces

Interfaces in the .NET Framework

- IComparable
- IEnumerable and IEnumerator
- IFormattable
- IList
- ICloneable
- IComponent
- IDataErrorInfo

Delegates

Definition

- Object oriented function pointers
- Multiple receivers
 - Each delegate has an invocation list
 - Thread-safe + and - operations
- Foundation for framework events

```
delegate void MouseEvent(int x, int y);  
  
delegate double Func(double x);  
  
Func MyFunc = new Func(Math.Sin);  
double x = MyFunc(1.0);
```

Delegates

Syntax

```
public delegate void MouseEvent(int x, int y);  
  
public delegate double Func(double x);  
  
Func MyFunc = new Func(Math.Sin);  
double x = MyFunc(1.0);
```

Custom Events

Creating, Firing and Handle

- Event signature and firing logic

```
public class MyClass
{
    public delegate void MyHandler(EventArgs e);
    public event MyHandler MyEvent;

    if (MyEvent != null) MyEvent(e);
}
```

- Handling the Event

```
public event EventHandler Click;
MyClass obj = new MyClass();
obj.MyEvent += new MyClass.MyHandler(MyFunc);
```

Predefined Events

System.EventHandler

- Define and register Event Handler

```
public class MyForm: Form
{
    Button okButton;

    public MyForm()
    {
        okButton = new Button(...);
        okButton.Caption = "OK";
        okButton.Click += new EventHandler(OkButtonClick);
    }

    void OkButtonClick(object sender, EventArgs e)
    {
        ShowMessage("You pressed the OK button");
    }
}
```

Operator Overloading

Definition

- Allows for the specification of an operator in the context of a class you have defined.
- Binary Operators
- Unary Operators
- Implicit type Conversions
- Explicit type conversions
- Only available in C#, not VB.NET

Operator Overloading

Syntax

```
public struct DBInt
{
    .....

    public static DBInt operator +(DBInt x, DBInt y) {...}

    public static implicit operator DBInt(int x) {...}
    public static explicit operator int(DBInt x) {...}
}
```

```
DBInt x = 123;
DBInt y = DBInt.Null;
DBInt z = x + y;
```

Operator Overloading

Binary and Unary Operators

- Binary Operators

– + - * / % & | ^ << >> == != > < >= <=

- Unary Operators

– + - ! ~ ++ --

Language Features

Versioning

- Overlooked in most languages
 - C++ and Java produce fragile base classes
 - Users unable to express versioning intent
- C# allows intent to be expressed
 - Methods are not virtual by default
 - C# keywords “virtual”, “override” and “new” provide context
- C# can't guarantee versioning
 - Can enable (e.g., explicit override)
 - Can encourage (e.g., smart defaults)

Language Features

Versioning

```
class Base                                // version 2
{
    public virtual void Foo() {
        Console.WriteLine("Base.Foo");
    }
}
```

```
class Derived: Base                       // version 2b
{
    public override void Foo() {
        base.Foo();
        Console.WriteLine("Derived.Foo");
    }
}
```

Summary

- C# builds on the .NET Framework component model
- New language with familiar structure
 - Easy to adopt for developers of C, C++, Java, and Visual Basic applications
- Fully object oriented
- Optimized for the .NET Framework

Thanks for your attention!



Object Oriented Programming