Static Polymorphism is achieved by using Function / Method Overloading

Operator Overloading

Dynamic Polymorphism :

Static Poly > The objects are linked to the methods that we can call at compile time

Dynamic Poly > The objects are linked to the methods that we can call at run time

Class A

{

Public void Get()

{}

}

Main()

{

A a = new A();

**a.Get(); The object a is linked to method Get () at compile time**

}

Class A

{

Public void Get()

{}

}

Class B : A

{

Public void Get()

{

}

}

}

Main()

{

A a = new A();

**a.Get(); The object a is linked to method Get () at compile time**

**B b = new B();**

**b.Get(); // It will call Get() method of B Class**

**// The object b is linked to method Get () at compile time**

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo3

{

class A

{

public void Get()

{

Console.WriteLine("Inside A Class Method");

}

}

class B: A

{

public void Get()

{

Console.WriteLine("Inside B Class Method");

}

}

class DynamicPolymorhismDemo

{

static void Main()

{

A a = new A();

a.Get();

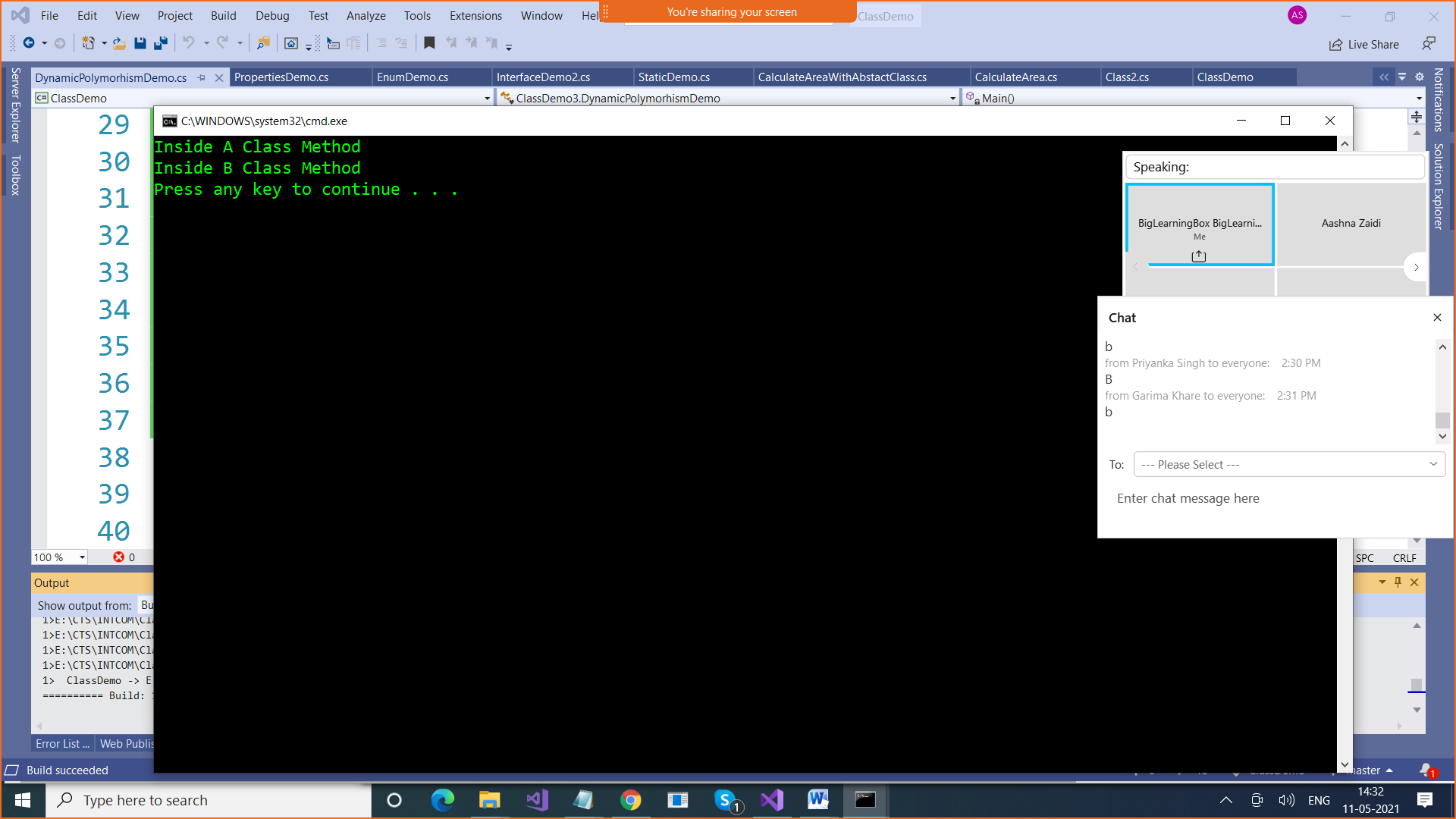
B b = new B();

b.Get();

}

}

}



using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo3

{

class A

{

public void Get()

{

Console.WriteLine("Inside A Class Method");

}

}

class B: A

{

public void Get()

{

Console.WriteLine("Inside B Class Method");

}

}

class DynamicPolymorhismDemo

{

static void Main()

{

A a = new A();

a.Get();

B b = new B();

b.Get();

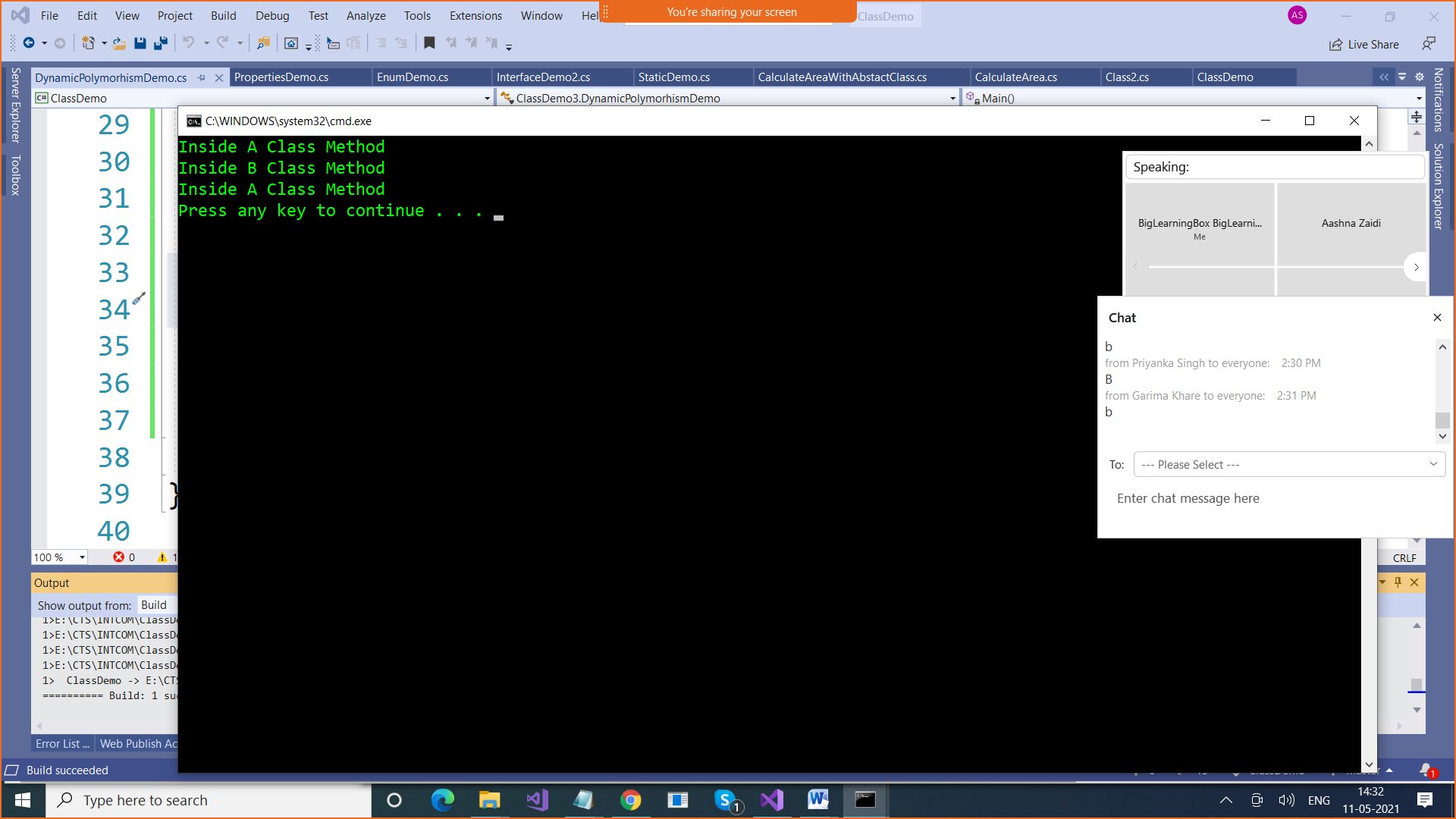
**a = b;**

**a.Get(); // Compile Time Linking . .Static Linking / Early Linking**

}

}

}



For Dynamic Linking , We use Virtual Functions

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo3

{

class A

{

**public virtual void Get()**

{

Console.WriteLine("Inside A Class Method");

}

}

class B: A

{

**public override void Get()**

**{**

**Console.WriteLine("Inside B Class Method");**

**}**

}

class DynamicPolymorhismDemo

{

static void Main()

{

A a = new A();

a.Get();

B b = new B();

b.Get();

**a = b;**

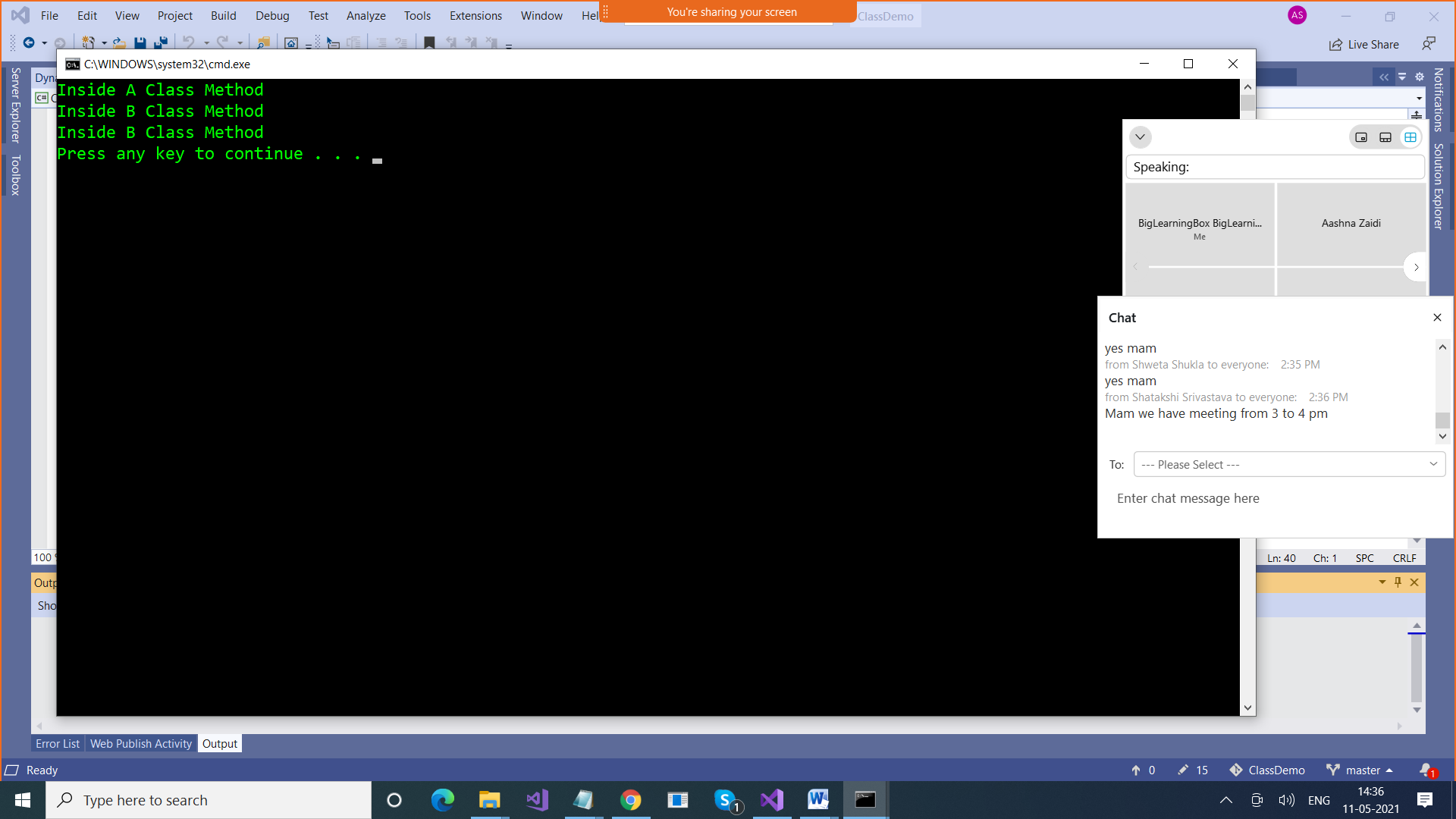
**a.Get(); // Because of Virtual Function, We achieved Run time Poly,**

**// which means objects are linked with methods at run time**

}

}

}



**Delegates**

**Pointers : Variables which point to other variables**

**Delegates : They are known as Function pointers**

**They point to functions at run time**

**How to use Delegates**

**1.Declare them**

**We have following 4 methods ,**

**Int add (int , int)**

**Int subtract (int , int)**

**Int multiply (int , int)**

**Int divide (int , int)**

**Declare Delegates**

**delegate <return\_type of the functions to which this delegate will point to> <delegatename> (parameters)**

**delegate int Del(int , int);**

**Delegates are of 2 types**

**SingleCast Delegates are Delegates which points to one function at a time**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

**// Delegate Declaration**

**delegate int MyDel(int x, int y);**

class number

{

public int add(int x, int y)

{

return x + y;

}

public int subtract(int x, int y)

{

return x - y;

}

public int product(int x, int y)

{

return x \* y;

}

public int divide(int x, int y)

{

return x / y;

}

}

class DelegateDemo

{

static void Main()

{

number number = new number();

**MyDel del = new MyDel(number.add);**

Console.WriteLine(del(10,5));

**del = new MyDel(number.subtract);**

Console.WriteLine(del(10, 5));

}

}

}

**MultiCast Delegate : A delegate points to more than 1 function at a time**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo4

{

**// Delegate Declaration**

**delegate void MyDel(int x, int y);**

class number

{

public void add(int x, int y)

{

Console.WriteLine("Sum is " + (x+y));

}

public void subtract(int x, int y)

{

Console.WriteLine("Difference is " + (x - y));

}

public void product(int x, int y)

{

Console.WriteLine("Product is " + (x \* y));

}

public void divide(int x, int y)

{

Console.WriteLine("Quotient is " + (x / y));

}

}

class MultiCastDelegateDemo

{

static void Main()

{

number number = new number();

MyDel del = new MyDel(number.add);

// MultiCast Delegate

**// += is used to add a method to a chain of methods**

del += new MyDel(number.subtract);

del += new MyDel(number.product);

del += new MyDel(number.divide);

del(20, 10);

**// -= is used to remove a method from chain of methods**

del -= new MyDel(number.add);

Console.WriteLine("Addition funct removed");

del(50, 10);

}

}

}

**Lambda Expressions / Anonymous Methods /LINQ**

