C# für Java Entwickler

Microsoft® Student Partners

Agenda

Präsentation

Zuhören
Grundkonzepte

Hands on Coding

Selber machen
Verstehen

Eure Fragen

Alles, was euch einfällt

About

Microsoft® Student Partners

About

Mich

Philipp Kühn
Masterstudent 2. Sem. Informatik
philipp.kuehn@studentpartners.de
www.cocktailsandcode.de

Student Partners

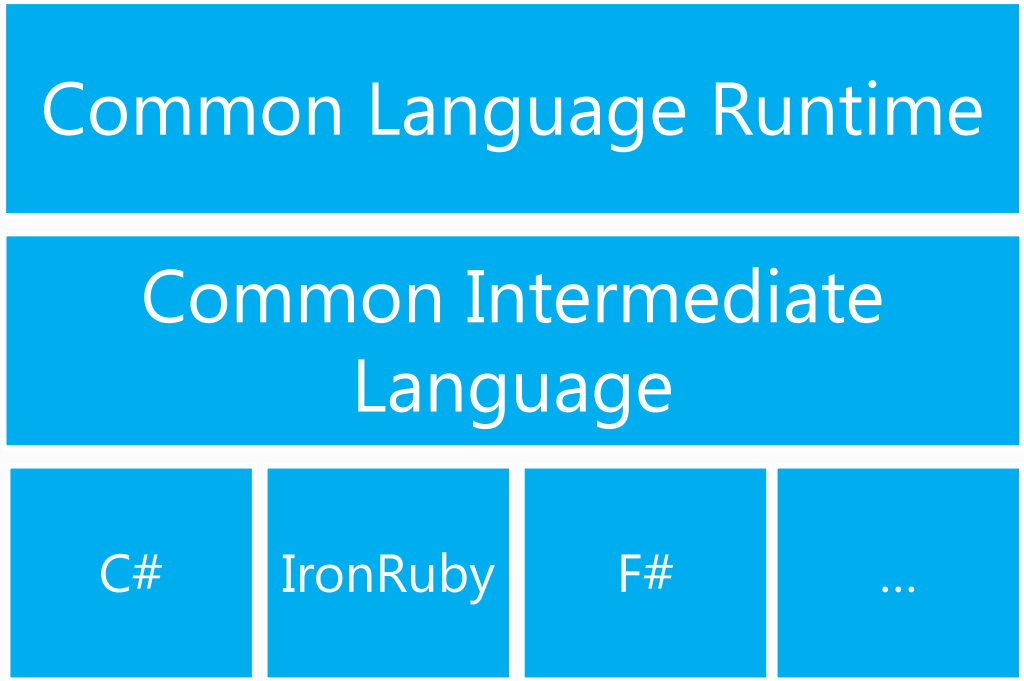
Programm von Microsoft
Fördern & Wissen vermitteln
Nicht bei Microsoft angestellt

Vergleich

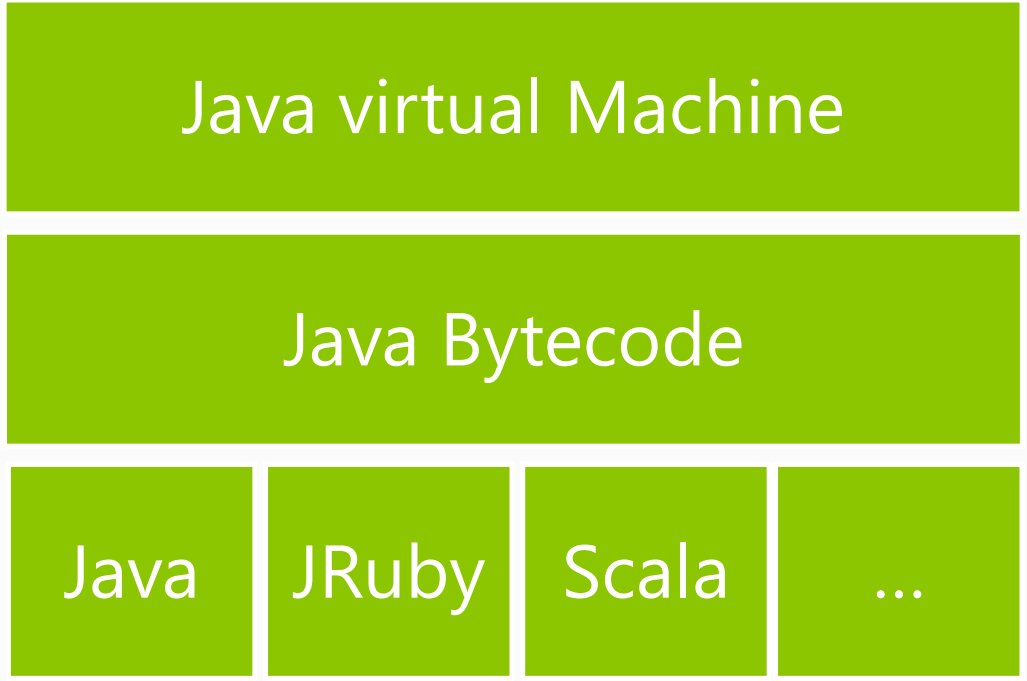
Microsoft® Student Partners

Plattform

.Net

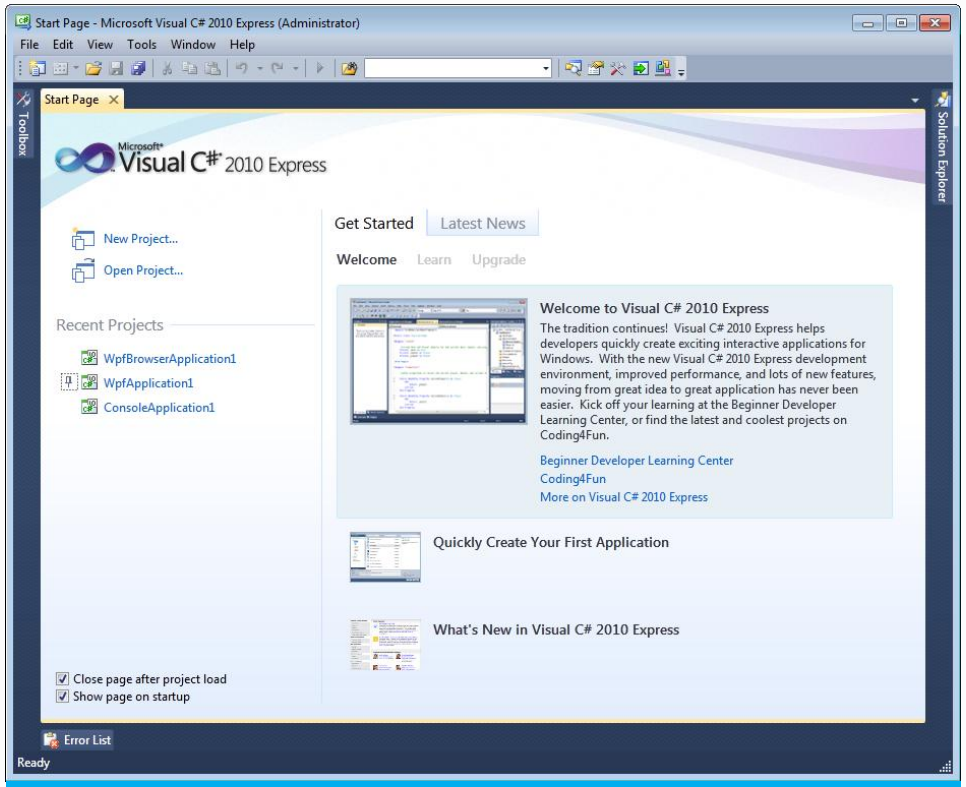


Java



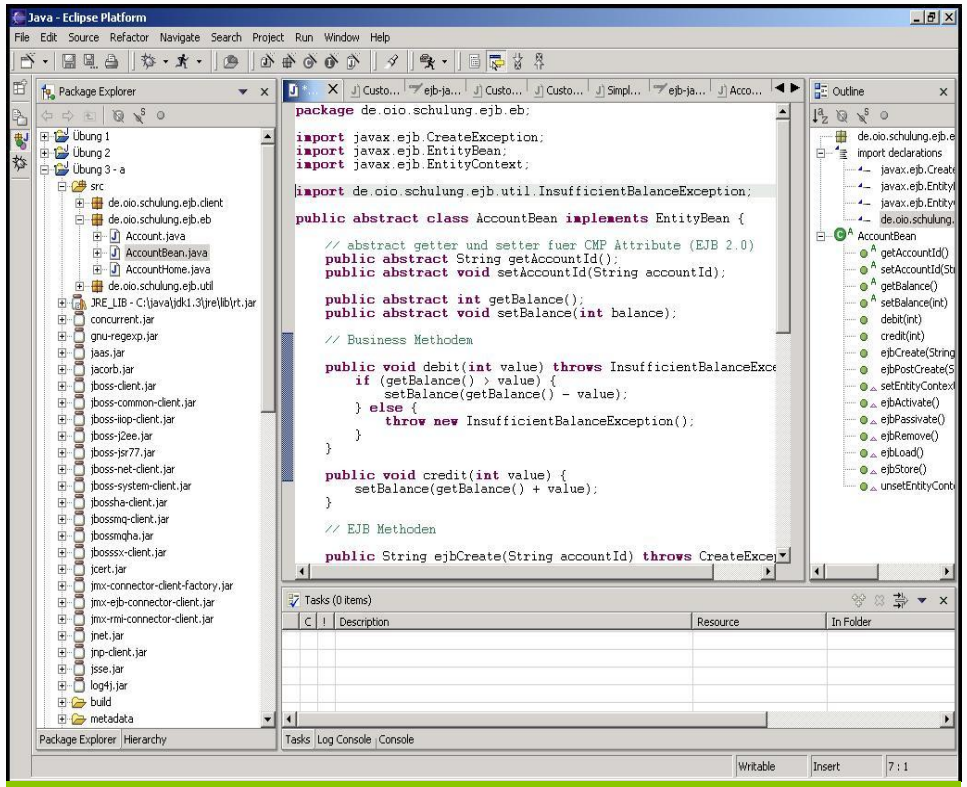
IDE

.Net



Visual Studio

Java



Eclipse

Namenskonventionen

Subjekt	.NET	Java
Namensraum/Paket	Pascal Case	Camel Case
Klassennamen	Pascal Case	Pascal Case
Methodennamen	Pascal Case	Pascal/Camel Case
Klassenvariablen	Camel Case	Camel Case
Properties, Events	Pascal Case	Pascal Case
Lokale Variablen	Camel Case	Camel Case
Schnittstellen	Beginnen mit I	Beginnen mit I

Pascal Case:

Alle Worte groß geschrieben, ohne Leerzeichen

Camel Case:

Wie Pascal Case, jedoch erster Buchstabe des ersten Wortes auch klein

Access Modifier

Subjekt	C#	Java
public	Alle	Alle
private	Enthaltender Typ	Enthaltender Typ
protected	Enthaltender Typ oder abgeleitete Typen	Package oder abgeleiteter Typ in anderem Package
internal	Enthaltende Assembly	-
protected internal	Enthaltende Assembly oder abgeleiteter Typ in anderer Assembly	-
(no modifier)	private	friendly (Package)

Werttypen

C#

```
int i = 1;
object o = i;
```

✓

Java

```
int i = 1;
object o = i;
```

✘

In C# sind alle Typen von object abgeleitet
int ist die Kurzform für Int32

Klassen

C#

```
using System;
using System.Collections.Generic;
using System.Linq; using System.Text;
namespace ConsoleApplication2
{
    /// <summary>
    /// Documentation is important!
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Java

```
package consoleApplication2;

import java.util.ArrayList;
import java.io.*;

/**
 * Documentation is important!
 */
public class Program {

        public static void main(String[] args) {
        }

}
```

Casting

C#

```
object o = new object();
int i = 0;
bool same = i is object;
o = i as object; // null, wenn nicht möglich
o = (object)i; // Exception, wenn nicht möglich
```

Java

```
Object o = new Object();
Integer i = 0;
boolean same = i instanceof Object;

o = (Object)i; // Exception, wenn nicht möglich
```

Schleifen

C#

```
for (int i = 0; i < max; i++)
{
    // do something
}
foreach (int i in list)
{
    // do something
}
```

Java

```
for(int i = 0; i < max; i++) {
        // do something
}

for(int i : list) {
        // do something
}
```

Getter, Setter

C#

```
private int x;

public int X
{
    get { return x; }
    set { x = value; }
}
```

Oder (Kurzform):

```
public int X { get; set; }
```

Java

```
private int x;

public int GetX() {
        return x;
}

public void SetX(int x) {
        this.x = x;
}
```

Vererbung

C#

```
class Program : BaseClass, ISomeInterface
```

Java

```
public class Program extends BaseClass implements ISomeInterface
```

Vererbung

C#

```
class A
{
    public virtual void foo()
    {
        Console.WriteLine("foo");
    }
    public void bar()
    {
        Console.WriteLine("bar");
    }
}
class B : A
{
    public override void foo()
    {
        Console.WriteLine("foo#2");
    }
     public new void bar()
    {
        Console.WriteLine("bar#2");
    }
}
class Programm
{
    static void Main(string[] args)
    {
        A a = new A();
        B b = new B();
        a.foo(); // foo
        a.bar(); // bar
        b.foo(); // foo#2
        b.bar(); // bar#2
        A c = (A)b;
        c.foo(); // foo#2
        c.bar(); // bar    }
}
```

Java

```
class A {
    public void foo() {
        System.out.println("foo");
    }
    public final void bar() {
            System.out.println("bar");
    }
}
class B extends A {
    public void foo() {
            System.out.println("foo#2");
    }
    /*public void bar() {
            System.out.println("bar#2");
    }*/
}
class Programm {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();

        a.foo(); // foo
        a.bar(); // bar
        b.foo(); // foo#2
        b.bar(); // bar
        A c = (A)b;
        c.foo(); // foo#2
        c.bar(); // bar
    }
}
```

Neues in C#

Microsoft® Student Partners

Delegates

```
delegate void Print(string s);
static void a(string s)
{
    Console.WriteLine("a " + s);
}
static void b(string s)
{
    Console.WriteLine("b " + s);
}
static void Display(Print printMethod)
{
    printMethod("someText");
}
static void Main(string[] args)
{
    Print callA = new Print(a);
    Print callB = new Print(b);
    Display(callA); // "a someText" es geht auch Display(a);
    Display(callB); // "b someText" es geht auch Display(b);
}
```

Events

```
static Timer myTimer;
static void Main(string[] args)
{
    myTimer = new Timer(1000);
    myTimer.Elapsed += new ElapsedEventHandler(myTimer_Elapsed);
    myTimer.Start();
}
static void myTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    Console.WriteLine("elapsed");
}
```

Lambda Expressions

```
Func<int, int> f = x => x + 1;
Console.WriteLine(f(5)); // 6

Action<int> a = x => { int y = x + 1; Console.WriteLine(y); };
a(5); // 6

Predicate<int> p = x => x == 5;
Console.WriteLine(p(5)); // true
```

var und dynamic

var

Type Interference
Wird zur Compile time überprüft
Eingeführt für LINQ

dynamic

Kompatibel zu allen Typen
Wird zur Run time überprüft
Eingeführt für IronRuby & IronPython

LINQ

```
var range = Enumerable.Range(0, 10);
var even = range.Where(x => x % 2 == 0);
var even2 = from x in range
            where x % 2 == 0
            select x;
var plusOne = range.Select(x => x + 1);
var countOdd = range.Count(x => x % 2 != 0);
```

Übung

Microsoft® Student Partners