C# versus Java

Hanspeter Mössenböck

Universität Linz
Institut für Praktische Informatik (Systemsoftware)
moessenboeck@ssw.uni-linz.ac.at

Auf einen Blick...

	Java	C#
Entstehungsjahr	1995	2001
Plattform	Java (Unix/Linux, Windows, MacOS, ...)	.NET (Windows, Linux)
Leitidee	Eine Sprache auf vielen Plattformen	Viele Sprachen auf einer Plattform (VB, C++, J#, Eiffel, Fortran, ...)
VM	Java-VM	Common Language Runtime
Zwischencode	Java-Bytecodes	Common Intermediate Lang.
JIT	per Methode	gesamt
Komponenten	Beans, EJB	Assemblies
Versionierung	-	ja
Applets	ja	-
Datenbanken	JDBC	ADO.NET
Web-Programmierung	Servlets, JSP	ASP.NET

Merkmale von C#

Sehr ähnlich zu Java

70% Java, 10% C++, 10% Visual Basic, 10% neu

Wie in Java

- Objektorientierung (einf. Vererbung)
- Interfaces
- Exceptions
- Threads
- Namespaces (wie Packages)
- Strenge Typprüfung
- Garbage Collection
- Reflection
- Dynamisches Laden von Code
- ...

Wie in C++

- (Operator) Overloading
- Zeigerarithmetik in Unsafe Code
- Einige syntaktische Details

Neue Features in C#

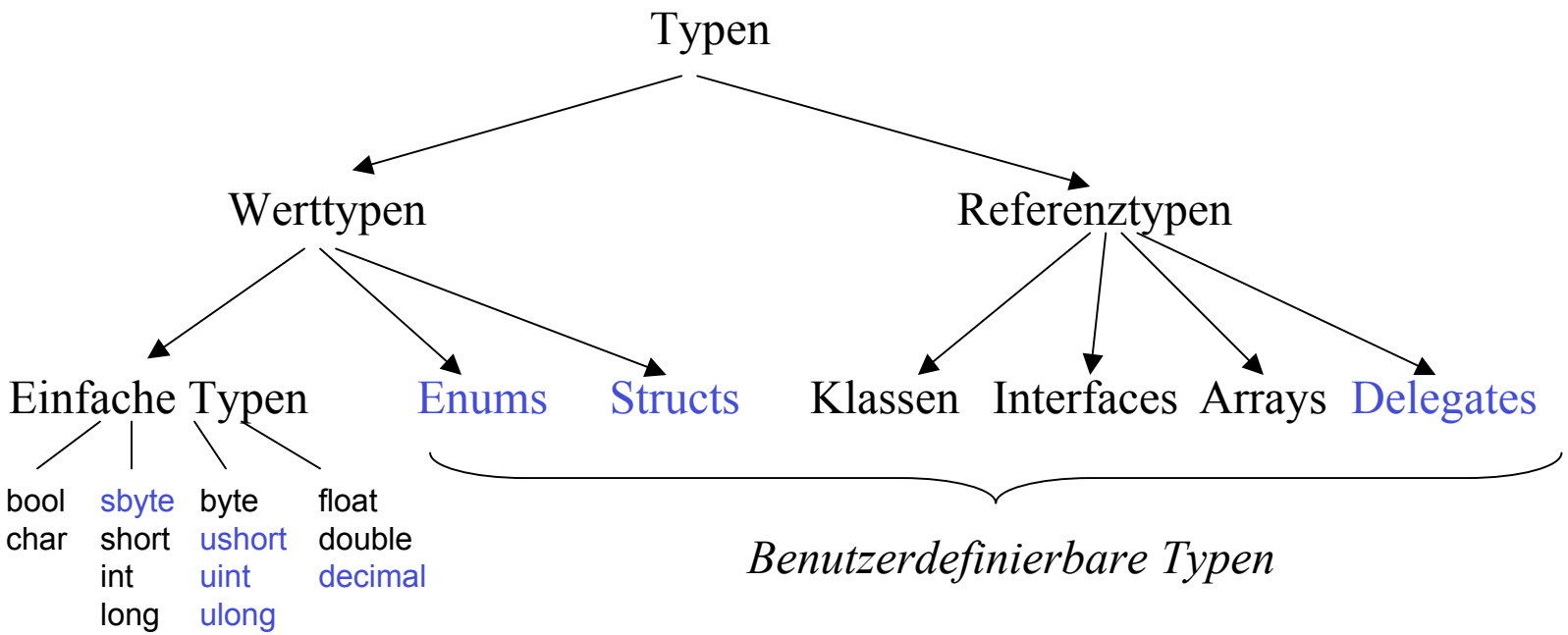
Wirklich neu (vs. Java)

- Referenzparameter
- Objekte am Stack (Structs)
- Blockmatrizen
- Enumerationen
- Uniformes Typsystem
- goto
- Systemnahes Programmieren
- Versionierung
- Attribute
- Kompatibel mit anderen .NET-Sprachen

"Syntactic Sugar"

- Komponentenunterstützung
  - Properties
  - Events
- Delegates
- Indexers
- Operator Overloading
- foreach-Iterator
- Boxing/Unboxing
- ...

Typsystem von C#



Alle Typen sind kompatibel mit *object*

- können *object*-Variablen zugewiesen werden
- verstehen *object*-Operationen

Boxing und Unboxing

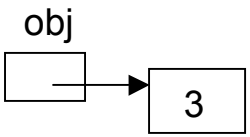
Auch Werttypen (int, struct, enum) sind zu *object* kompatibel!

Boxing

Bei der Zuweisung

```
object obj = 3;
```

wird der Wert 3 in ein Heap-Objekt eingepackt



Java

```
Object obj = new Integer(3);
```

Unboxing

Bei der Zuweisung

```
int x = (int) obj;
```

wird der eingepackte int-Wert wieder ausgepackt

```
int x = ((Integer)obj).intValue();
```

Klassen und Vererbung

C#

```
class A {
   private int x;
   public A(int x) { this.x = x; }
}

class B : A, I1, I2 {
   private int y;

   public int Bar() { ...}

   public B(int x, int y) : base(x) {
      this.y = y;
   }
}
```

Java

```
class A {
   private int x;
   public A(int x) { this.x = x; }
}

class B extends A implements I1, I2 {
   private int y;

   public int Bar() { ...}

   public B(int x, int y) {
      super(x);
      this.y = y;
   }
}
```

Überschreiben von Methoden

C#

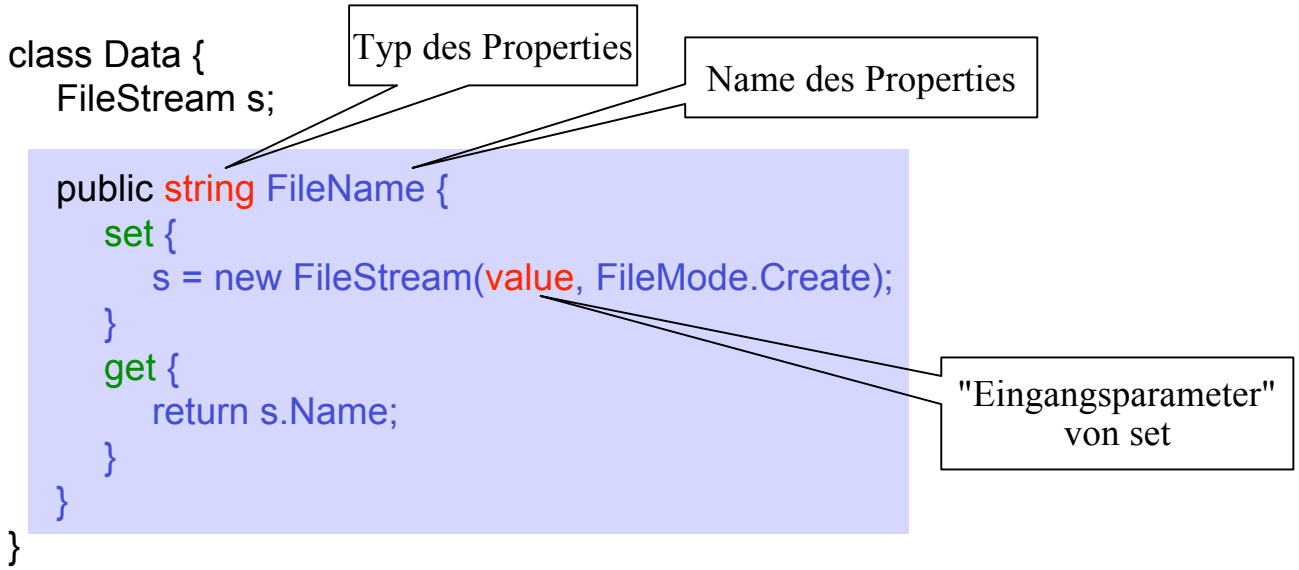
```
class A {
    ...
    public virtual void Foo() { ...}
}

class B : A {
    ...
    public override void Foo() {
        base.Foo();
        ...
    }
}
```

Java

```
class A {
    ...
    public void Foo() { ...}
}

class B extends A {
    ...
    public void Foo() {
        super.Foo();
        ...
    }
}
```

Properties

Syntaktische Kurzform für get/set-Methoden




```
class Data {
    FileStream s;

    public string FileName {
        set {
            s = new FileStream(value, FileMode.Create);
        }
        get {
            return s.Name;
        }
    }
}
```

Wird wie ein Feld benutzt ("smart fields")

```
Data d = new Data();

d.FileName = "myFile.txt";      // ruft d.set("myFile.txt") auf
string s = d.FileName;          // ruft d.get() auf
```

Properties (Fortsetzung)

get oder set kann fehlen

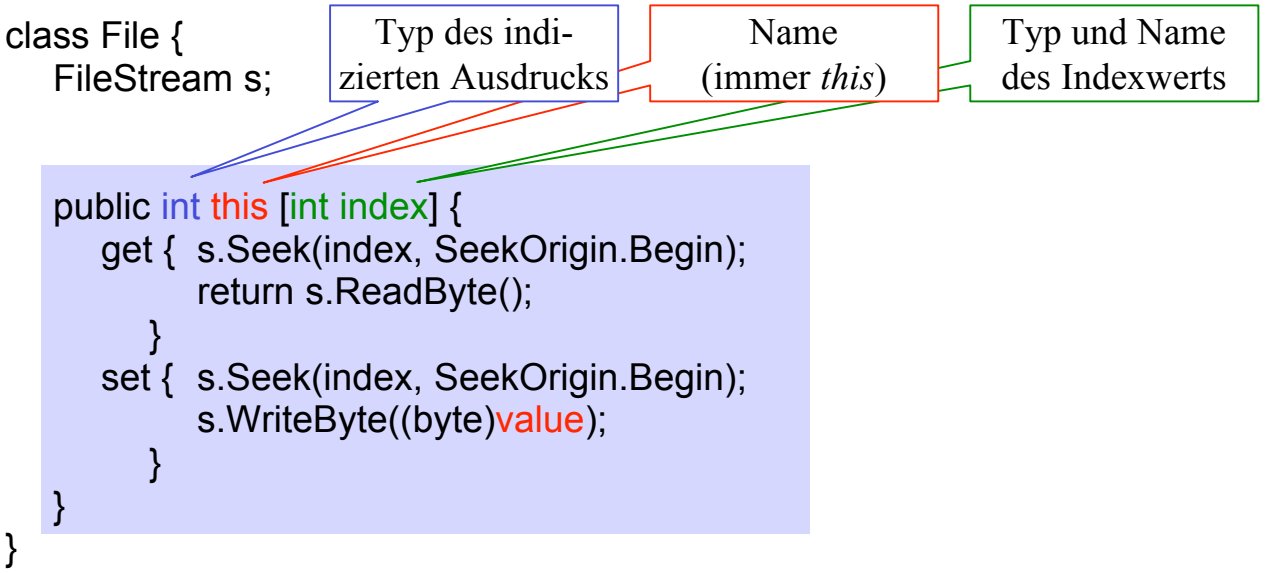
```
class Account {
    long balance;

    public long Balance {
        get { return balance; }
    }
}

x = account.Balance;        // ok
account.Balance = ...;      // verboten
```

Nutzen von Properties

- read-only und write-only-Daten möglich.
- Validierung beim Zugriff möglich.
- Benutzersicht und Implementierung der Daten können verschieden sein.

Indexer

Programmierbarer Operator zum Indizieren einer Folge (Collection)




```
class File {
    FileStream s;

    public int this [int index] {
        get {  s.Seek(index, SeekOrigin.Begin);
               return s.ReadByte();
            }
        set {  s.Seek(index, SeekOrigin.Begin);
               s.WriteByte((byte)value);
            }
    }
}
```

Benutzung

```
File f = ...;
int x = f[10];          // ruft f.get(10)
f[10] = 'A';            // ruft f.set(10, 'A')
```

C# ist oft griffiger als Java

C#

```
Hashtable tab = new Hashtable();
...
tab["John"] = 123;
...
int x = (int) tab["John"];
```

```
String s = ...;
...
foreach (char ch in s)
    process(ch);
```

Java

```
Hashtable tab = new Hashtable();
...
tab.put("John", new Integer(123));
...
int x = ((Integer) tab.get("John")).intValue();
```

```
String s = ..;
...
for (int i = 0; i < s.length(); i++)
    process(s.charAt(i));
```

Ausnahmebehandlung (Exceptions)

C#

```
try {
    ...
    ... throw myException;
    ...
} catch (E1 e) {
    ...
} catch (E2) {
    ...
} catch {
    ...
} finally {
    ...
}
```

Exception
- SystemException
  - IOException
    - FileNotFoundException
    - ...
  - ...
- ApplicationException

Java

```
try {
    ...
    ... throw myException;
    ...
} catch (E1 e) {
    ...
} catch (E2 e) {
    ...
} catch (Throwable e) {
    ...
} finally {
    ...
}
```

Throwable
- Exception
  - IOException
    - FileNotFoundException
    - ...
  - ...
  - RuntimeException
- Error

Keine Checked Exceptions in C#

C#

```
void Foo() {
   try {
      if (...)
         throw new E1();
      else
         throw new E2();
      ...
   } catch (E1 e) {
      ...
   }
}
```

Java

```
void Foo() throws E2 {
   try {
      if (...)
         throw new E1();
      else
         throw new E2();
      ...
   } catch (E1 e) {
      ...
   }
}
```

Keine Checked Exceptions

Wenn keiner der Rufer E2 abfängt, bricht das Programm mit einem Laufzeitfehler ab.

Checked Exceptions

Ausnahmen müssen abgefangen oder mit throws im Methodenkopf spezifiziert werden.

Checked Exceptions

Pro

+ Dokumentation für den Programmierer
+ Zwingt dazu, jeden Fehlerfall zu bedenken

Kontra

- Mühsam, wenn Ausnahme weitergereicht wird

```
void P() {
   try {
      ... Q(); ...
   } catch (E e) {...}
}
```

```
void Q() throws E {
   ... R(); ...
}
```

```
void R() throws E {
   ... S(); ...
}
```

```
void S() throws E {
   ...
   throw new E();
}
```

- Verführt zu folgendem "Hack"

```
try {
   ...
} catch (E e) {}
```

Fehler wird verschluckt!

Delegate = Methodentyp

Deklaration eines Delegate-Typs

```
delegate void Notifier (string sender);     // normale Methodensignatur
                                            // mit Schlüsselwort delegate
```

Deklaration einer Delegate-Variablen

```
Notifier notify;
```

Zuweisung einer Methode an eine Delegate-Variable

```
void SayHello(string sender) {
    Console.WriteLine("Hello " + sender);
}

notify = new Notifier(SayHello);
```

```
void SayGoodBye(string sender) {
    Console.WriteLine("Good bye " + sender);
}

notify = new Notifier(SayGoodBye);
```

Aufruf der Delegate-Variablen

```
notify("Max");                  // Aufruf von SayHello("Max") => "Hello Max"
```

Multicast-Delegates

Delegate-Variable kann mehrere Werte gleichzeitig aufnehmen

```
Notifier notify;
notify = new Notifier(SayHello);
notify += new Notifier(SayGoodBye);
```

```
notify("Max");                    // "Hello Max"
                                  // "Good bye Max"
```

```
notify -= new Notifier(SayHello);
```

```
notify("Max");                    // "Good bye Max"
```

Threads

C#

```
void P() {
    ... thread actions ...
}

Thread t = new Thread(new ThreadStart(P));
t.Start();
```

- Beliebige Methode kann als Thread gestartet werden
- Keine Unterklasse von *Thread* nötig

Java

```
class MyThread extends Thread {
    public void run() {
        ... thread actions ...
    }
}

Thread t = new MyThread();
t.start();
```

- Thread-Aktionen müssen in einer *run*-Methode stecken
- Unterklasse von *Thread* nötig, bzw. Implementierung von *Runnable*

Thread-Synchronisation

C#

```
class Buffer {
    int[] data;

    public void Put(int x) {
        lock(this) {
            ...
        }
    }

    [MethodImpl(MethodImplOptions.Synchronized)]
    public int Get() {
        ...
    }
}
```

Java

```
class Buffer {
    int[] data;

    public void put(int x) {
        synchronized(this) {
            ...
        }
    }

    public synchronized int get() {
        ...
    }
}
```

Monitor.Wait(this);	wait();
Monitor.Pulse(this);	notify();
Monitor.PulseAll(this);	notifyAll();

Attribute

Benutzerdefinierte Informationen über Programmelemente

- Können an Typen, Felder, Methoden, etc. angehängt werden.
- Werden als Metadaten gespeichert.
- Können zur Laufzeit abgefragt werden.
- Sind Unterklassen von *System.Attribute*.

Beispiel

```
[Serializable]
class C {...}                          // macht C serialisierbar
```

Mehrfache Attribute zuweisbar

```
[Serializable] [Obsolete]
class C {...}

[Serializable, Obsolete]
class C {...}
```

Beispiel: [Conditional]-Attribut

Für bedingte Aufrufe von Methoden

```
#define debug                                   // Präprozessor-Anweisung

class C {

   [Conditional("debug")]
   static void Assert (bool ok, string errorMsg) {
      if (!ok) {
         Console.WriteString(errorMsg);
         System.Environment.Exit(0);
      }
   }

   static void Main (string[] arg) {
      Assert(arg.Length > 0, "no arguments specified");
      Assert(arg[0] == "...", "invalid argument");
      ...
   }
}
```

Assert wird nur aufgerufen, wenn *debug* definiert wurde.

J# - Java unter .NET

- Vollständige Java-Syntax (JDK 1.1.4), ähnlich J++
- Delegates wie in C#
- Zugriff auf Properties anderer .NET-Programme

  statt: `label.Text = "xxx";` Methoden: `label.set_Text("xxx");`

  `text = label.Text;` `text = label.get_Text();`
- Unterstützt die meisten Java-Bibliotheken (nicht RMI, JNI)
- Unterstützt die meisten .NET-Bibliotheken (ASP.NET, ADO.NET, Windows Forms)
- Kann unter Visual Studio.NET verwendet werden
- Erzeugt keine Java-Bytecodes sondern .NET-Assemblies
  => nicht portabel, keine Applets

Zusammenfassung

C#	Java
mächtiger Structs, Referenzparameter, Attribute, ...	kleiner und einfacher
bequemer und griffiger Indexer, Boxing, foreach, ...	größere Verbreitung
flexibler erlaubt Systemprogrammierung	strikter Checked Exceptions, kein Unsafe Code
besser unter Windows	portabler