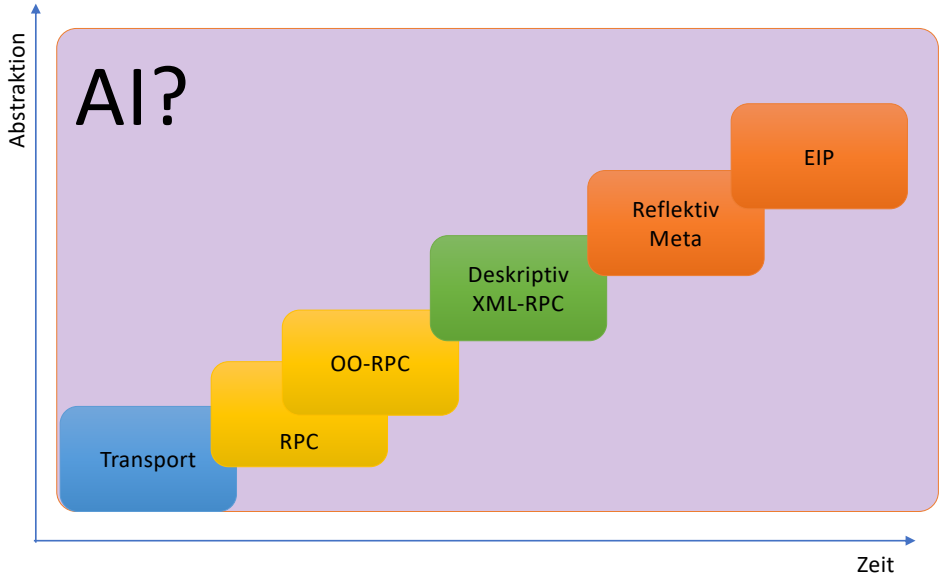

Software
Architectures
for Enterprises
SA4E
Peter Sturm
Universität Trier
Winter 2024

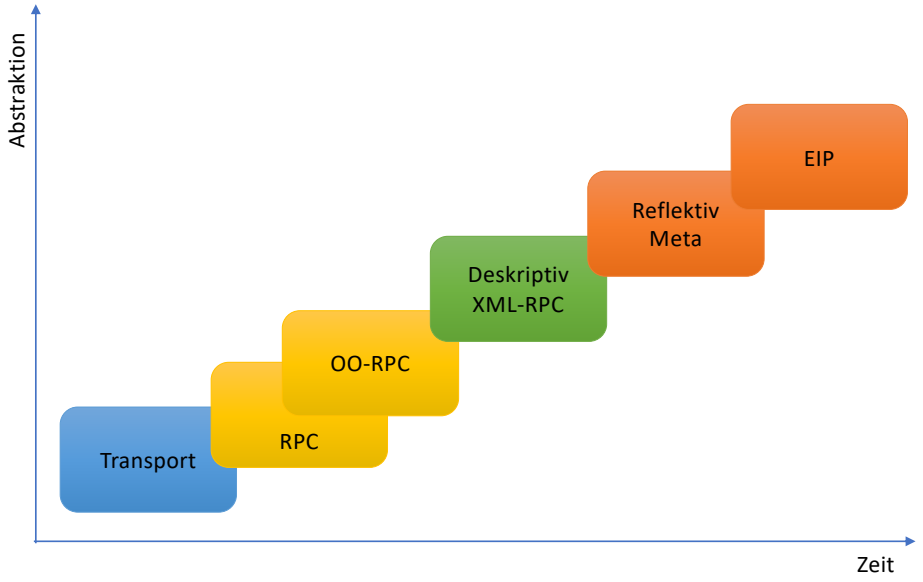

4. Reflektiv
2


4. Sprachen

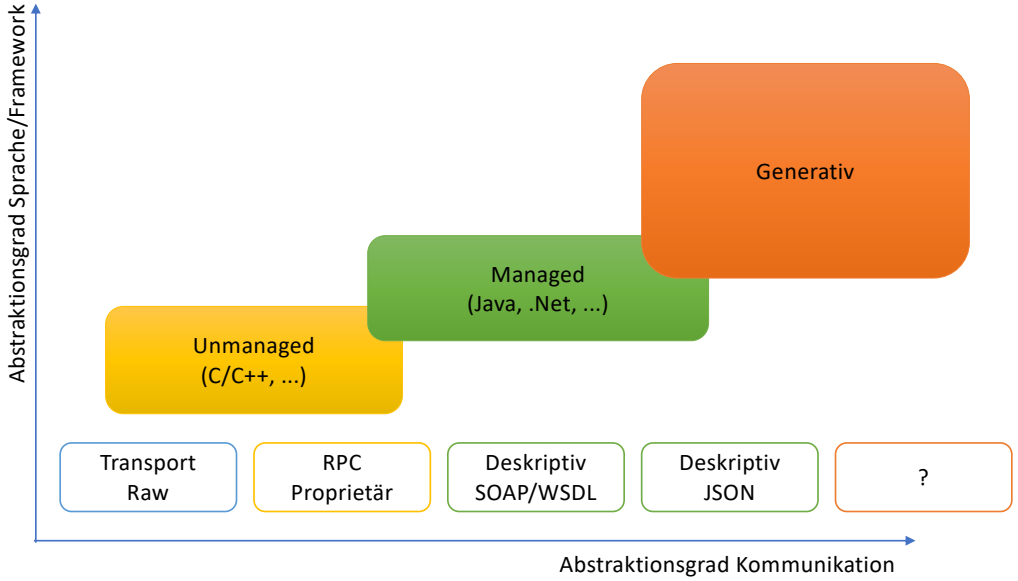
und weiter ...


Abstraktion
AI?
EIP
Reflektiv
Meta
Deskriptiv
XML-RPC
OO-RPC
RPC
Transport
Zeit

und weiter ...



und weiter ...



Es war einmal …

- TCP/IP
- RPC
- OO-RPC
- CORBA
- COM
- WebServices
- RESTful APIs

„Steinzeit“

- Hauptsache es geht
- Marginaler Abstraktionsgrad
  - Zahlen statt symbolische Namen
  - Begrenzte kommunizierbare Datentypen
- Ressourcen-limitierte Computer
- De Facto Standards

Technologien sind unsterblich


• Schicht über Schicht
• Write-Only Code
• …

Clean Slate Approach

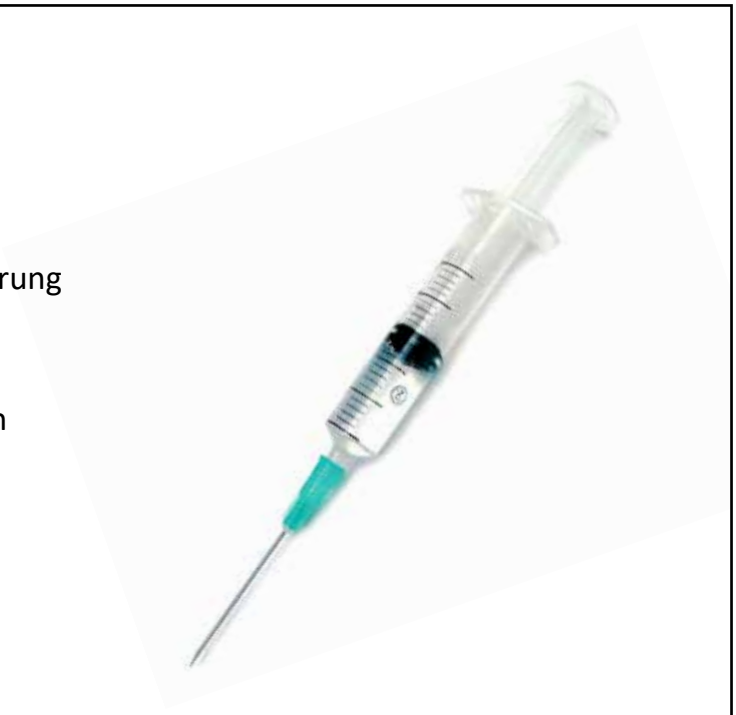
- Java und .NET (und HTML5?)
- Aus der Vergangenheit lernen
  - Keine Mehrfachvererbung
  - Garbage Collection
  - …
- Schnelle Computer und schnelle Netze

Wir lieben Compiler

- Hohe Abstraktionen
- Mehr Bequemlichkeit
  - Höhere Qualität
  - Weniger Fehler
- Code-Generierung

Neue Konzepte

- Generische Programmierung
- Closures
- Attribute / Annotationen
- Dependency Injection
- ...



Umsorgen / Bemuttern

- Adressierung
- Persistenz
- Sicherheit
- Administration
- Verifikation
- Performanz
- Versionspflege

Zeitgeist

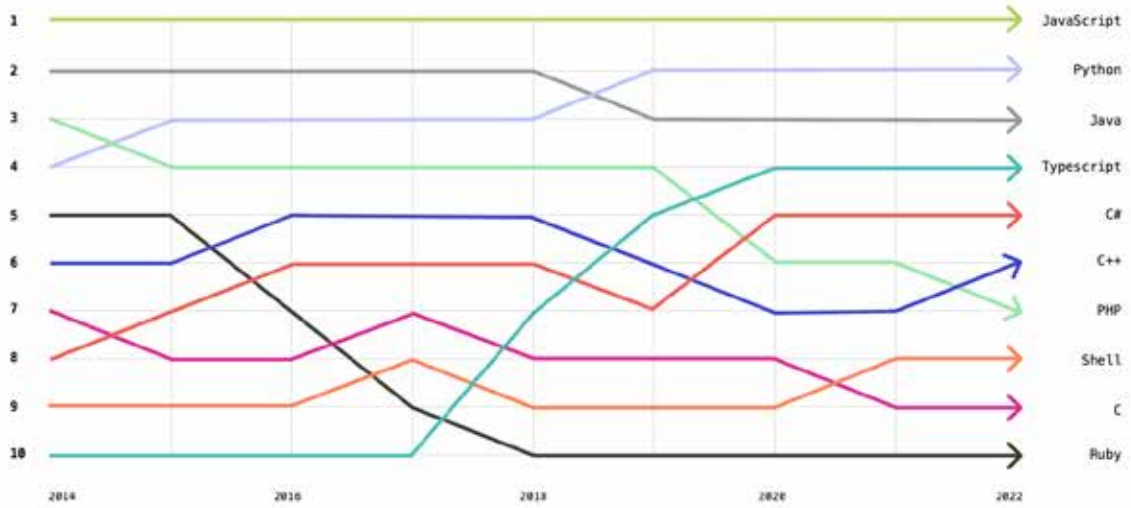
- Aktuelle Konzepte und Trends fließen ein
- Manches wird kalter Kaffee
- Problemfeld „Trägheit“

Nächste Schritte

“Managed”

- Eine der wenigen Revolutionen
  - Viel Neues in kurzer Zeit
- Bekannteste Vertreter
  - Java / JVM
  - C# / CLR
  - Python
  - Swift

github 2022



24

https://octoverse.github.com/2022/top-programming-languages#:~:text=Top%20languages%20used%20in%202022,place%20year%2Dover%2Dyear.

Octoverse 2024

https://github.blog/news-insights/octoverse/octoverse-2024/#the-most-popular-programming-languages

25

TIOBE (Dezember 2020)

Dec 2020	Dec 2019	Change	Programming Language	Ratings	Change
1	2	^	C	16.46%	+0.40%
2	1	v	Java	12.53%	-4.72%
3	3		Python	12.21%	+1.90%
4	4		C++	6.91%	+0.71%
5	5		C#	4.20%	-0.60%
6	6		Visual Basic	3.92%	-0.83%
7	7		JavaScript	2.35%	+0.26%
8	8		PHP	2.12%	+0.07%
9	16	^^	R	1.60%	+0.60%
10	9	v	SQL	1.53%	-0.31%
11	22	^^	Groovy	1.53%	+0.69%
12	14	^	Assembly language	1.35%	+0.28%
13	10	v	Swift	1.22%	-0.27%
14	20	^^	Perl	1.20%	+0.30%
15	11	vv	Ruby	1.16%	-0.15%
16	15	v	Go	1.14%	+0.15%
17	17		MATLAB	1.10%	+0.12%
18	12	vv	Delphi/Object Pascal	0.87%	-0.41%
19	13	vv	Objective-C	0.81%	-0.39%
20	24	^^	PL/SQL	0.78%	+0.04%

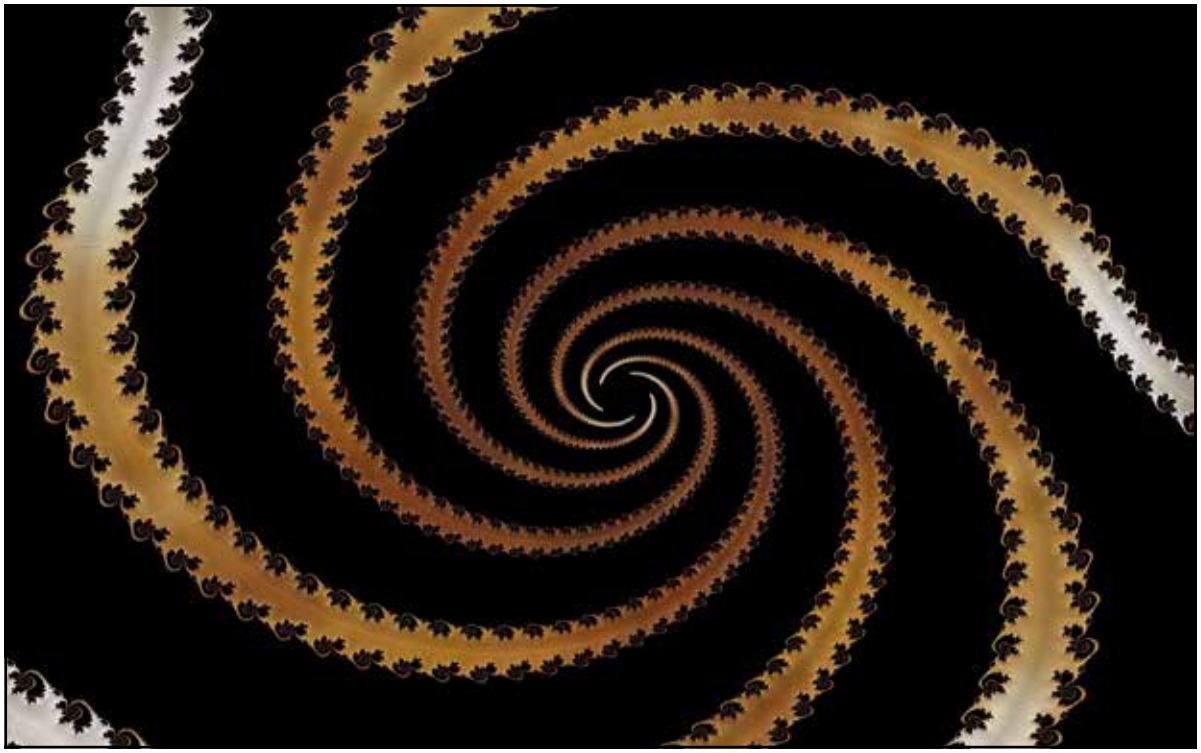
26

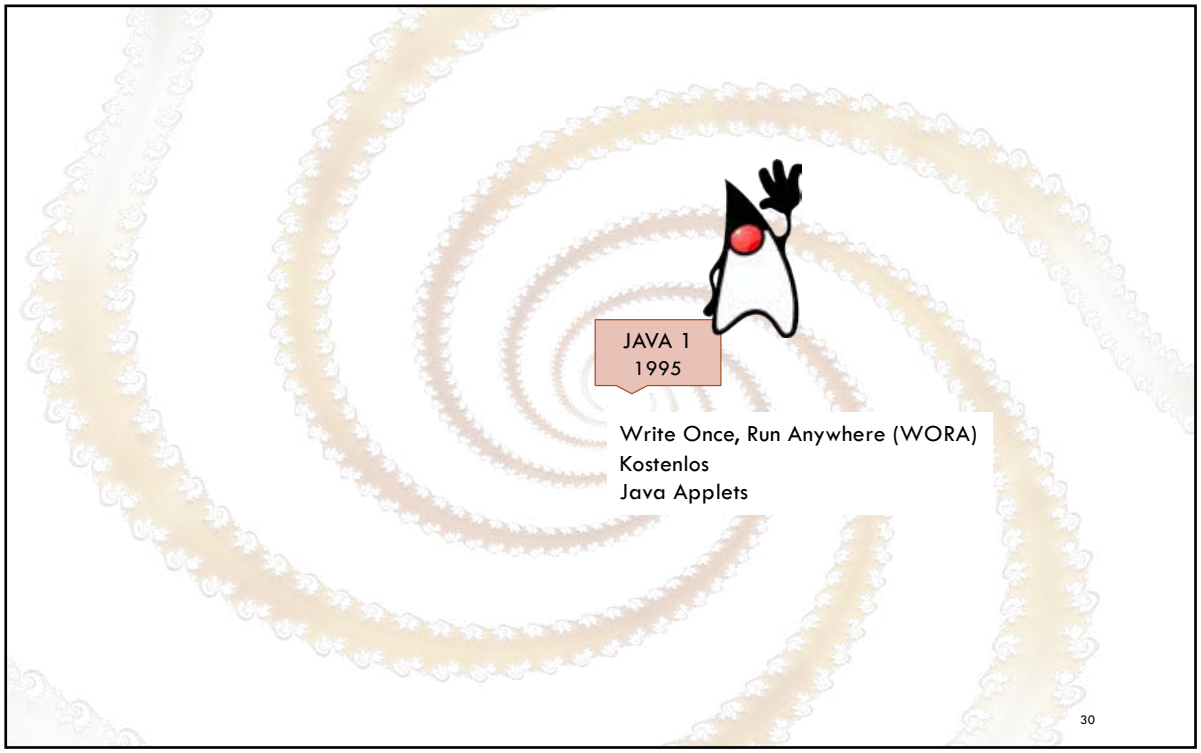
TIOBE

About us · Knowledge · News · Coding Standards · TIOBE Index · Contact

Products · Quality Models · Markets · Schedule a demo

Nov 2024	Nov 2023	Change	Programming Language	Ratings	Change
1	1		Python	22.85%	+8.69%
2	3	^	C++	10.64%	+0.29%
3	4	^	Java	9.60%	+1.26%
4	2	v	C	9.01%	-2.76%
5	5		C#	4.98%	-2.67%
6	6		JavaScript	3.71%	+0.50%
7	13	^^	Go	2.35%	+1.16%
8	12	^^	Fortran	1.97%	+0.67%
9	8	v	Visual Basic	1.95%	-0.15%
10	9	v	SQL	1.94%	+0.05%
11	16	^^	Delphi/Object Pascal	1.48%	+0.33%
12	7	vv	PHP	1.47%	-0.82%
13	14	^	MATLAB	1.28%	+0.12%
14	20	^^	Rust	1.17%	+0.20%




JAVA 1
1995
Write Once, Run Anywhere (WORA)
Kostenlos
Java Applets
30

JAVA 1
1995

JAVA 1.1
1997

JavaBeans
Reflection
RMI
De facto Standard (ECMA-Versuch)

31

JAVA 1
1995

JAVA 1.1
1997

JAVA 1.2
1998

Enterprise Beans (J2EE)
J2ME
Collections
IDL

PLATZ HIRSCH

32

Neue JVM
RMI für CORBA
JNDI

JAVA 1.3
2000

JAVA 1
1995

JAVA 1.1
1997

JAVA 1.2
1998

33

regexp
Non-Blocking IO
XML, XSLT
Web Start

.NET 1.0
02

JAVA 1.4
2002

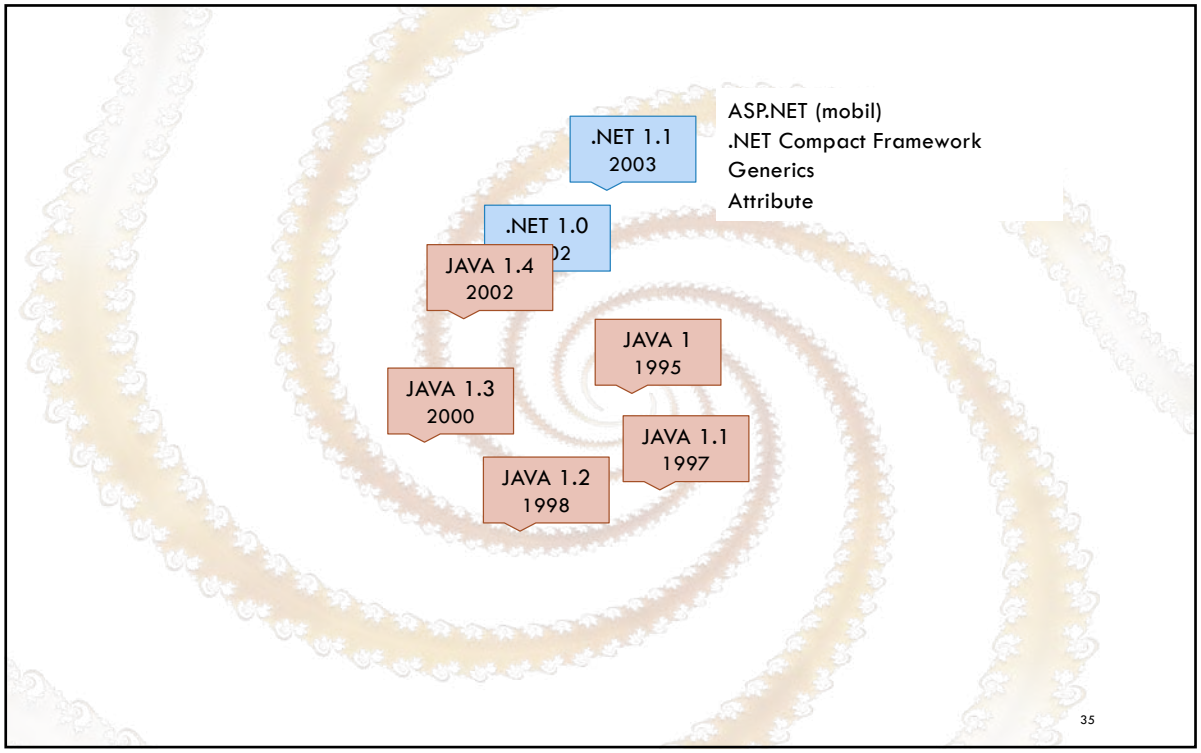
JAVA 1
1995

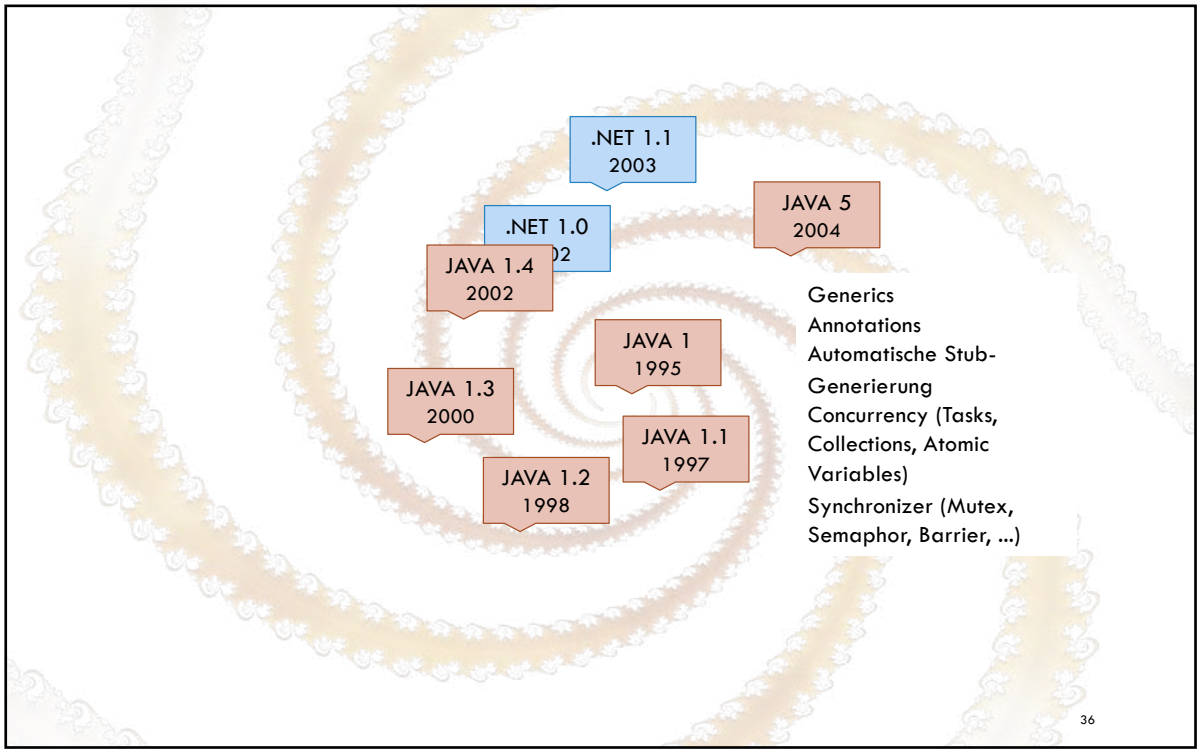
JAVA 1.3
2000

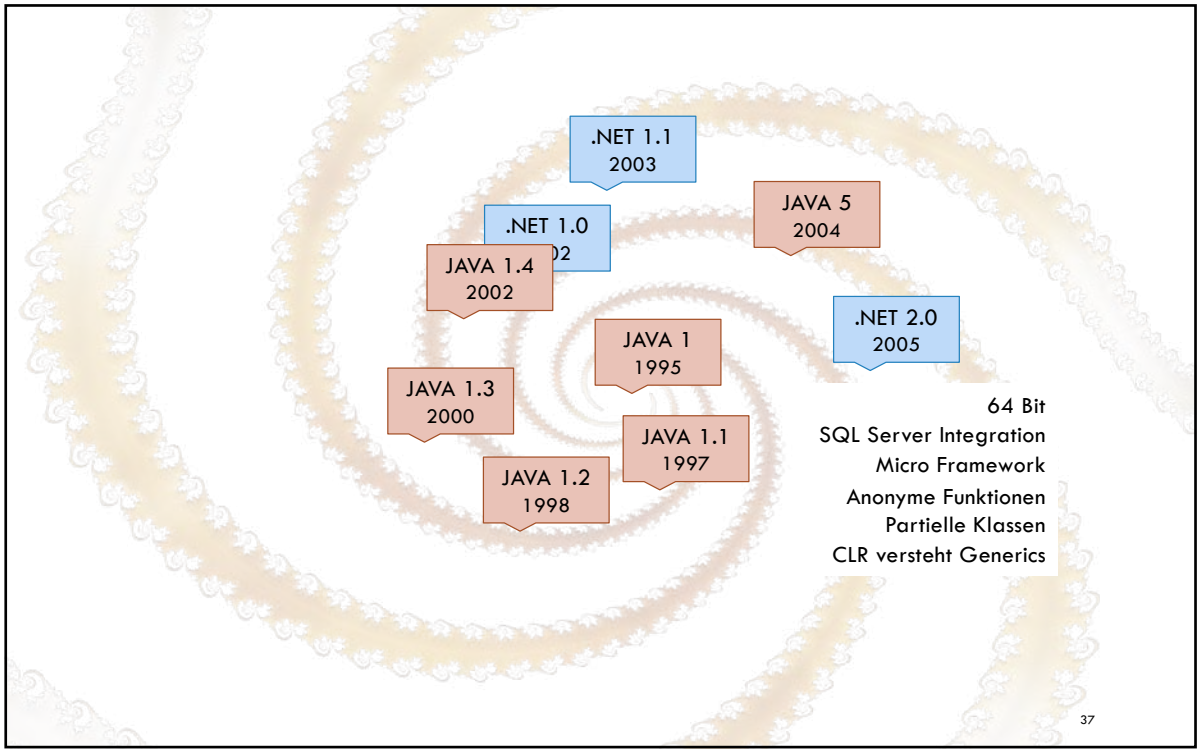
JAVA 1.1
1997

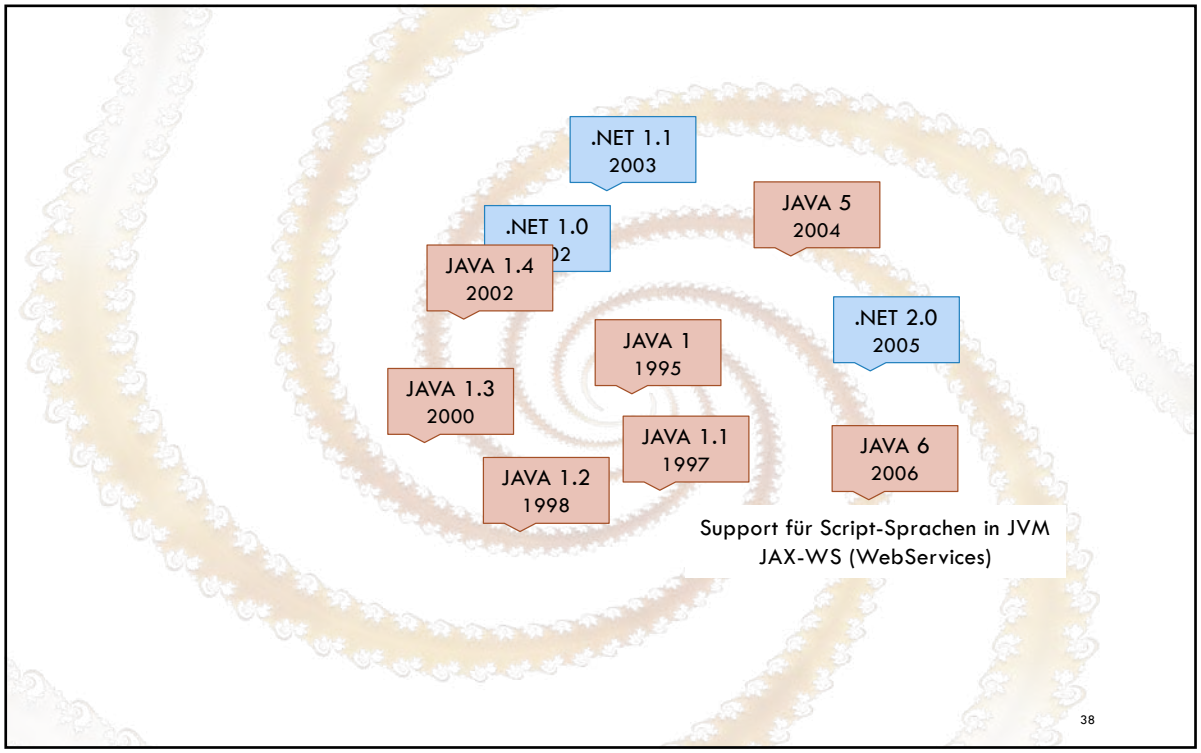
JAVA 1.2
1998

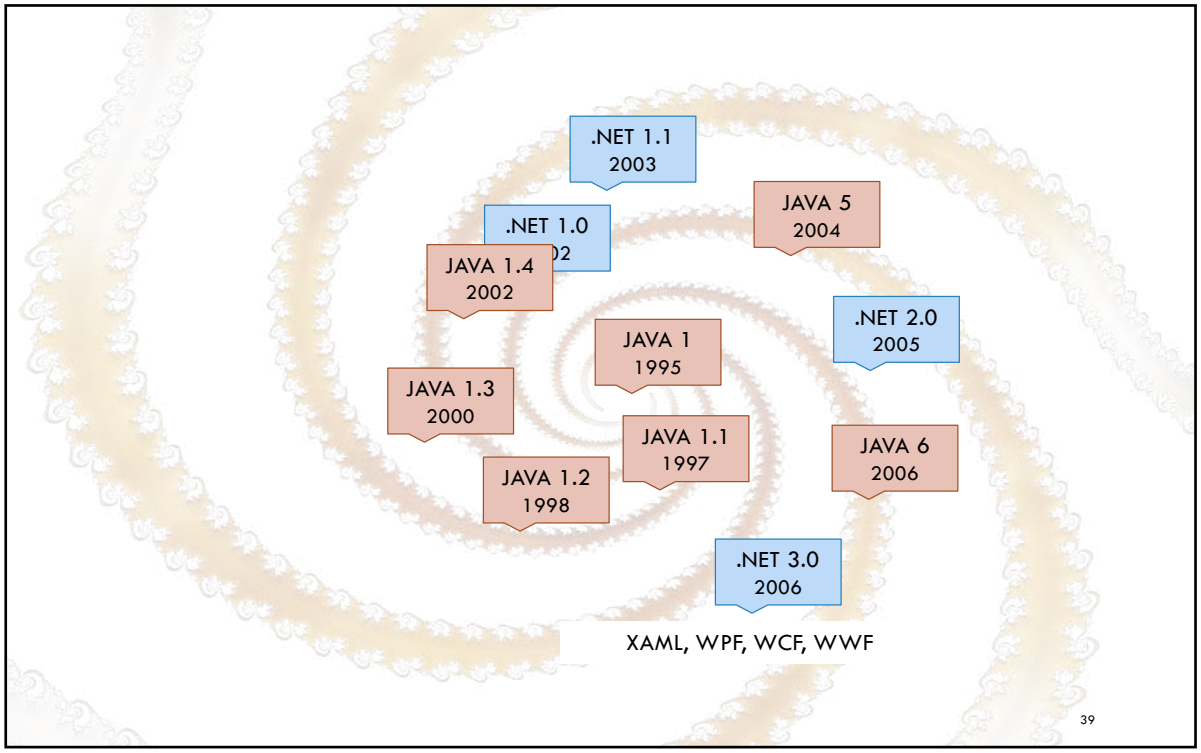
34

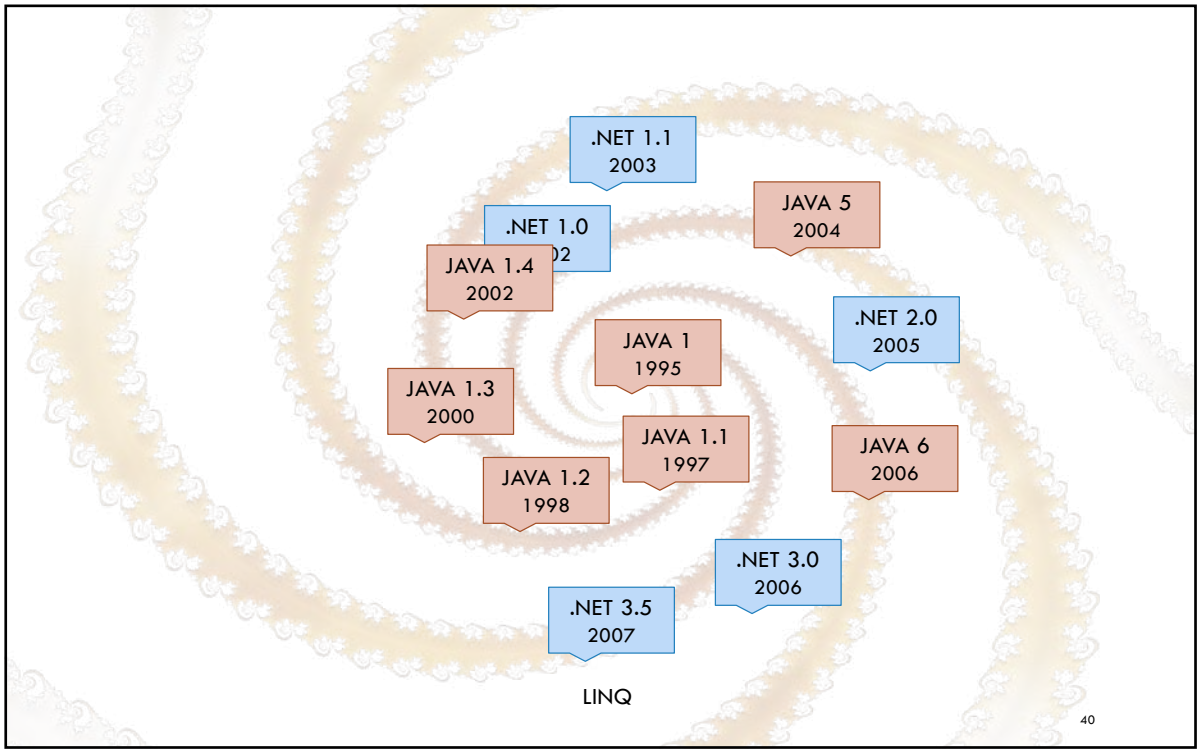

.NET 1.1
2003
ASP.NET (mobil)
.NET Compact Framework
Generics
Attribute
.NET 1.0
JAVA 1.4
2002
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 1.2
1998
35

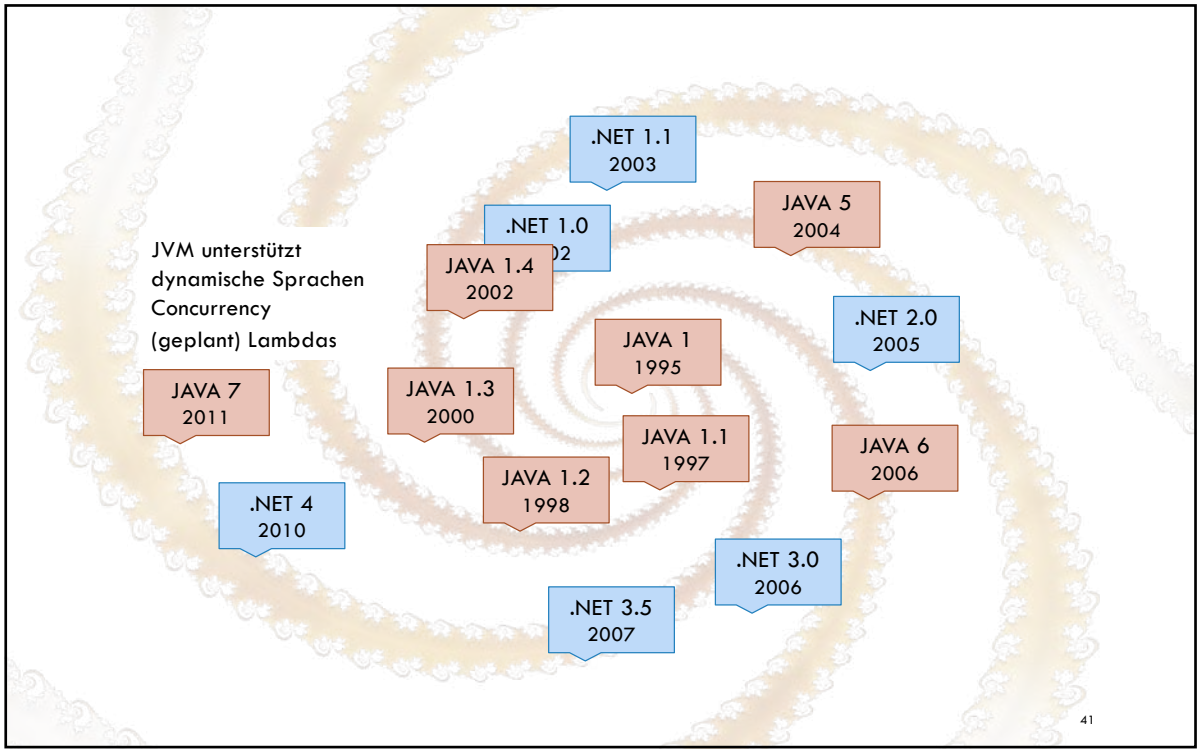

.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
Generics
Annotations
Automatische Stub-Generierung
Concurrency (Tasks, Collections, Atomic Variables)
Synchronizer (Mutex, Semaphor, Barrier, ...)
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 1.2
1998
36

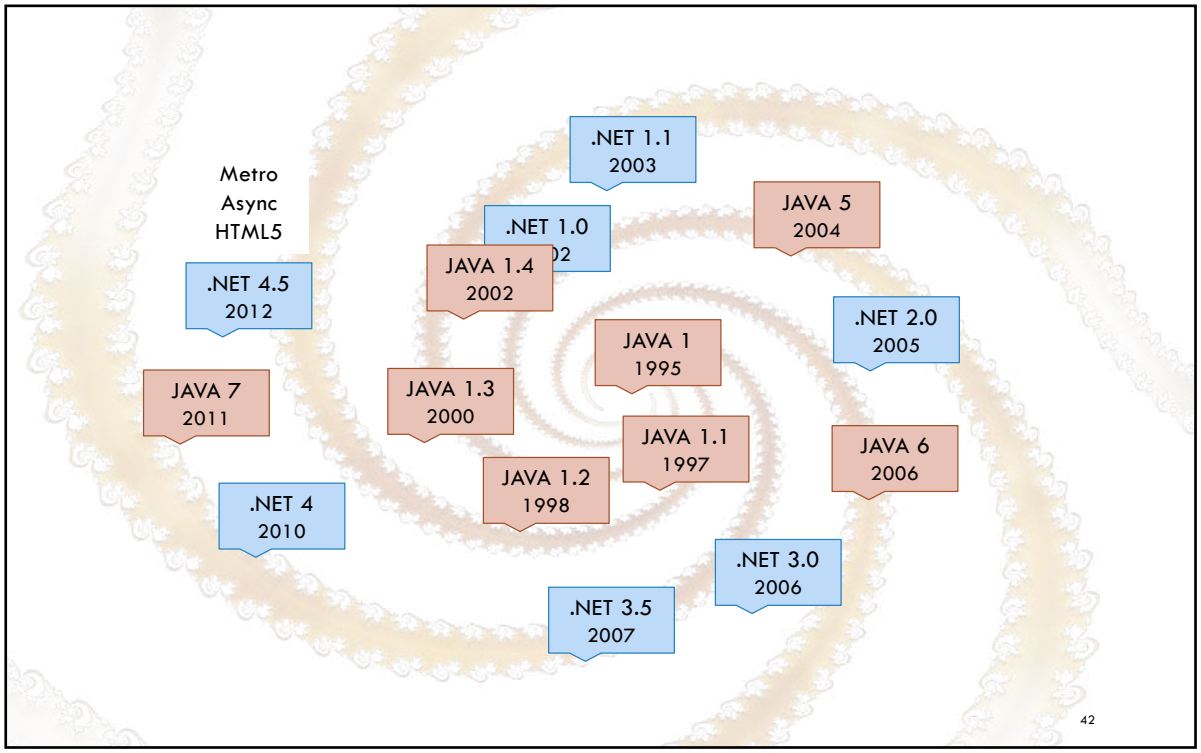

.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 2.0
2005
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 1.2
1998
64 Bit
SQL Server Integration
Micro Framework
Anonyme Funktionen
Partielle Klassen
CLR versteht Generics
37


.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 2.0
2005
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
Support für Script-Sprachen in JVM
JAX-WS (WebServices)
38

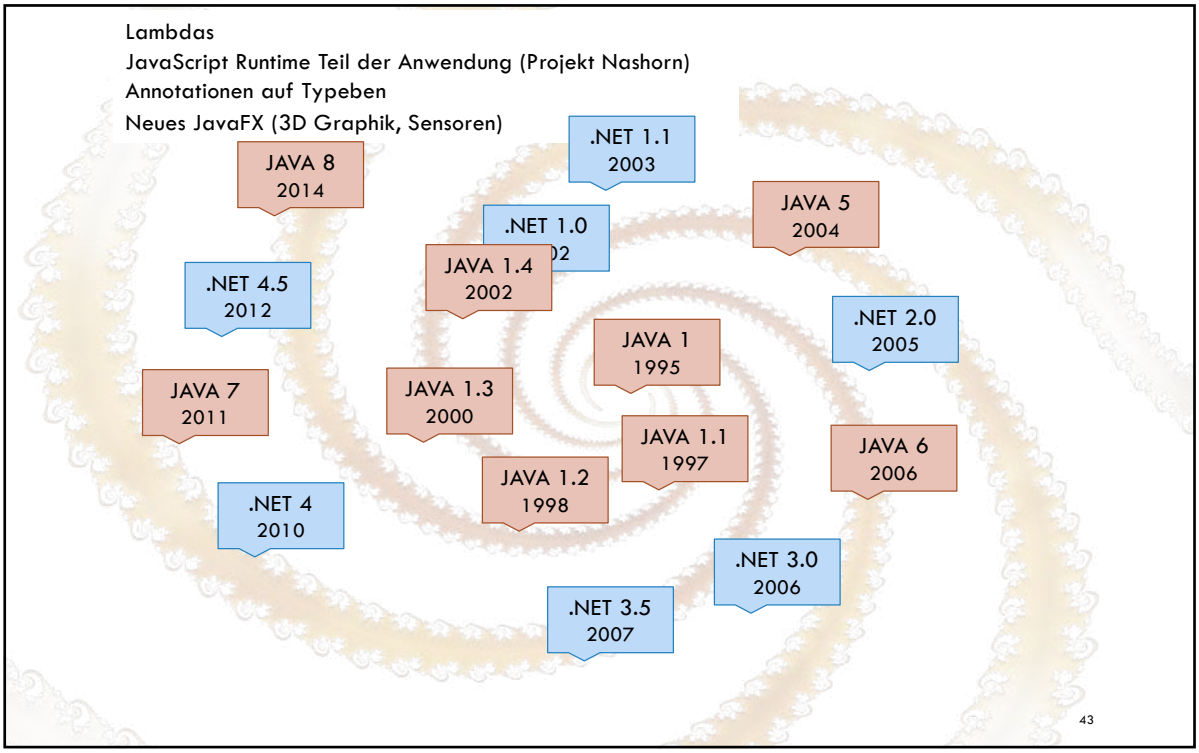

.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 2.0
2005
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
.NET 3.0
2006
XAML, WPF, WCF, WWF
39

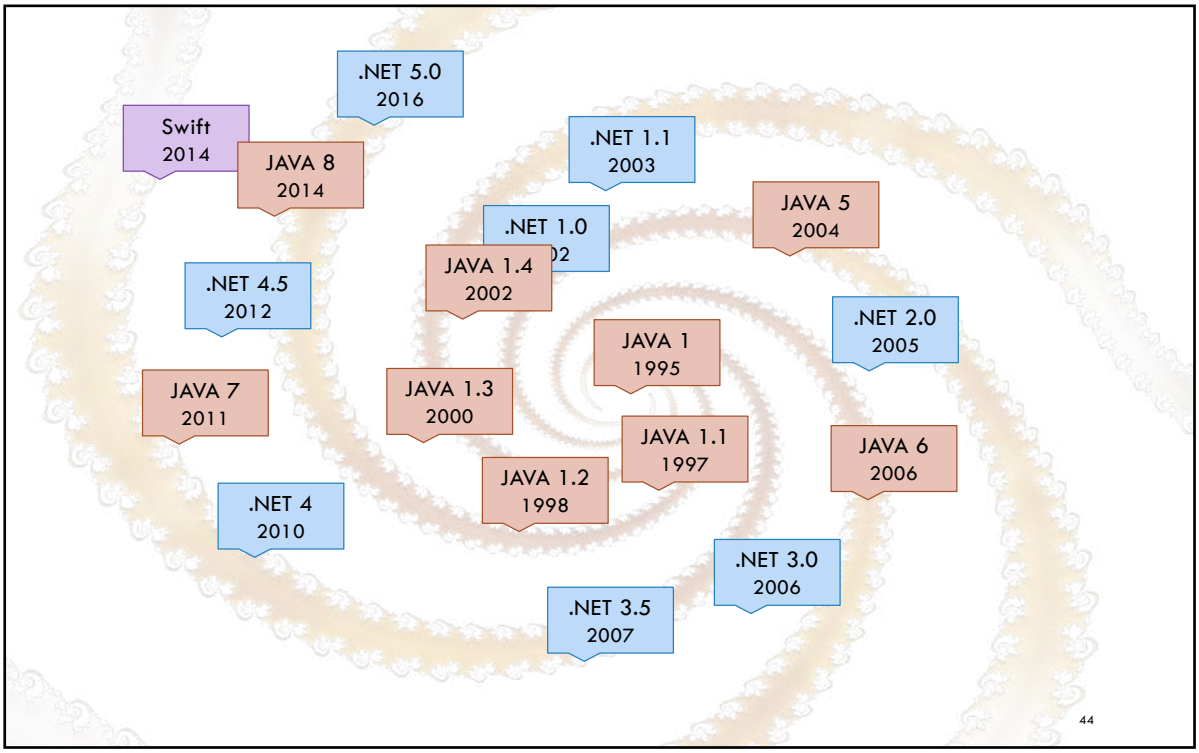

.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 2.0
2005
JAVA 1
1995
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
.NET 3.0
2006
.NET 3.5
2007
LINQ
40

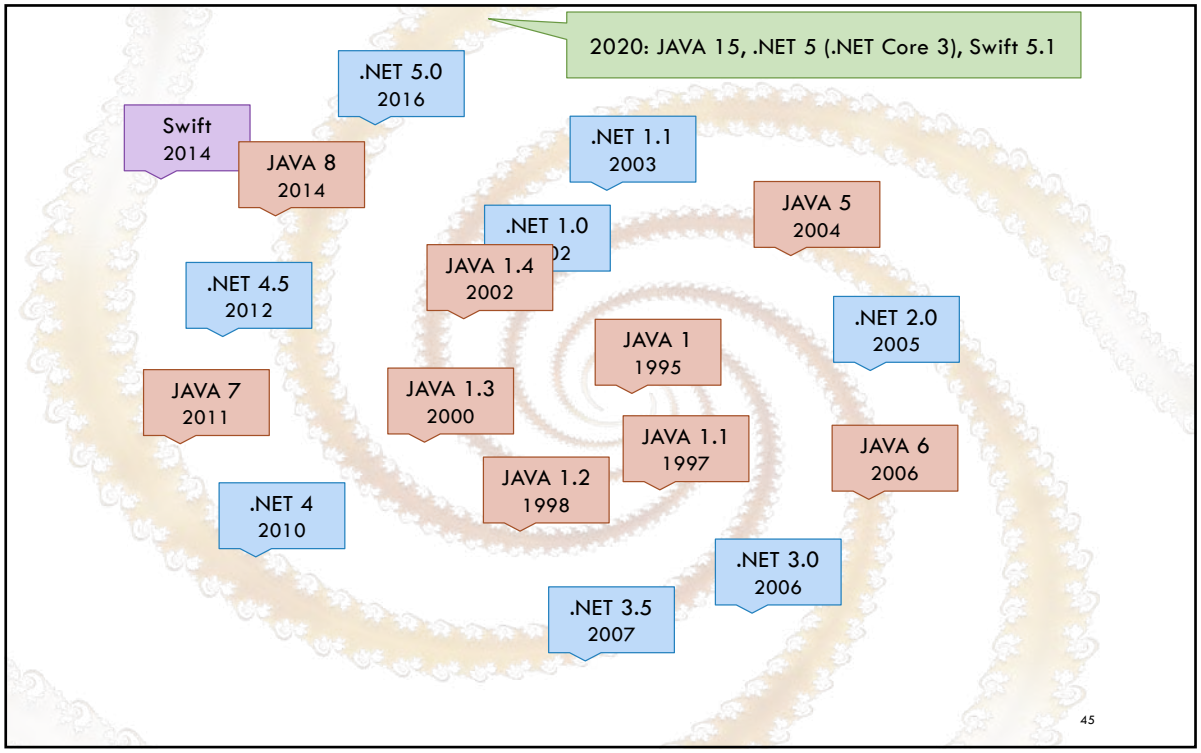

JVM unterstützt
dynamische Sprachen
Concurrency
(geplant) Lambdas
.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 2.0
2005
JAVA 1
1995
JAVA 7
2011
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
.NET 4
2010
.NET 3.0
2006
.NET 3.5
2007
41

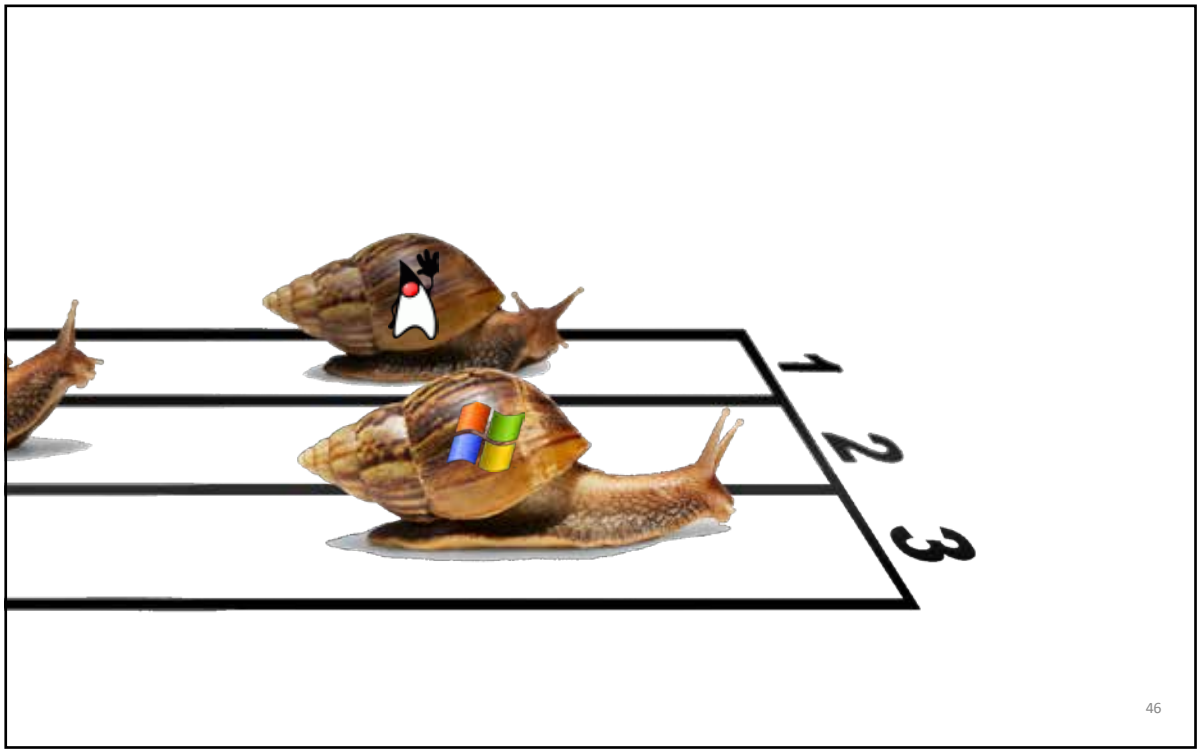

Metro
Async
HTML5
.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 4.5
2012
.NET 2.0
2005
JAVA 1
1995
JAVA 7
2011
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
.NET 4
2010
.NET 3.0
2006
.NET 3.5
2007
42

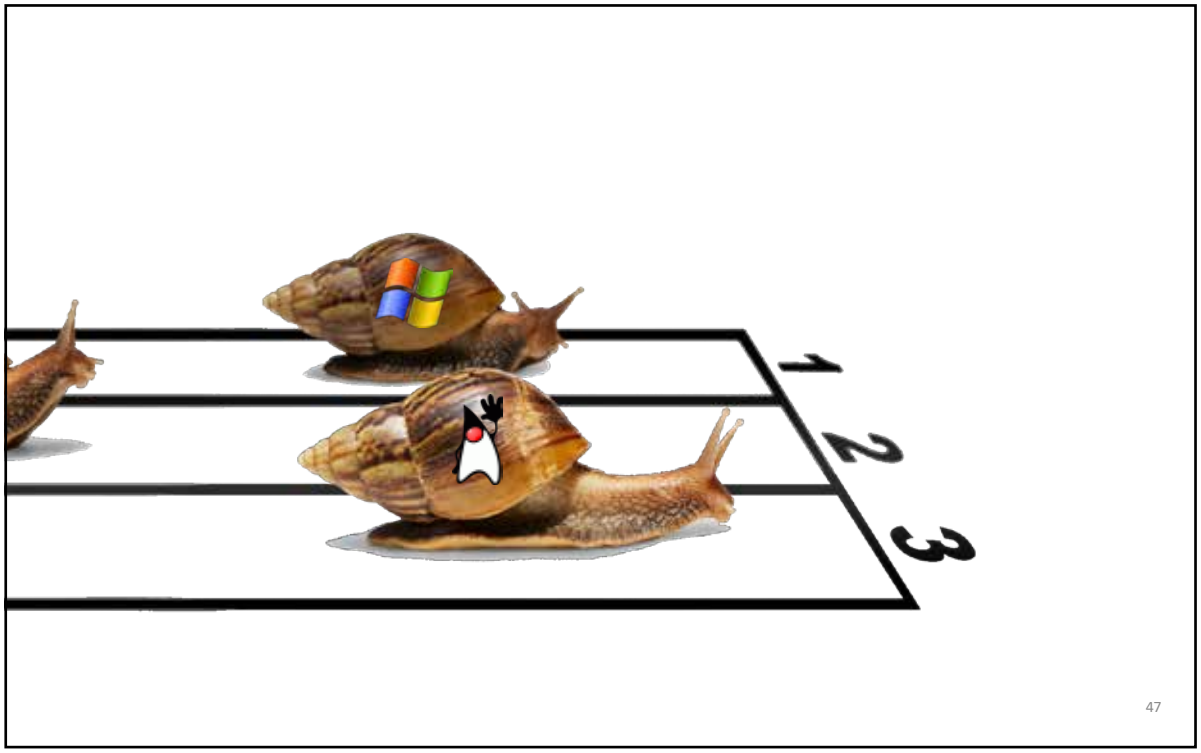
Lambdas
JavaScript Runtime Teil der Anwendung (Projekt Nashorn)
Annotationen auf Typeben
Neues JavaFX (3D Graphik, Sensoren)






2020: JAVA 15, .NET 5 (.NET Core 3), Swift 5.1
.NET 5.0
2016
Swift
2014
JAVA 8
2014
.NET 1.1
2003
JAVA 5
2004
.NET 1.0
JAVA 1.4
2002
.NET 4.5
2012
.NET 2.0
2005
JAVA 1
1995
JAVA 7
2011
JAVA 1.3
2000
JAVA 1.1
1997
JAVA 6
2006
JAVA 1.2
1998
.NET 4
2010
.NET 3.0
2006
.NET 3.5
2007
45


1
2
3
46


1
2
3
47


Syntactical Sugar

Zeitgeist

- Wachsende Bedeutung von XML und JSON
  - Integration relevanter Bibliotheken (Serialisierung, u.a.)
  - Webservices
  - XAML -> WCF, WPF und WWF
- Attribut-orientierte Programmierung
- Generische Programmierung
- Dynamische statt statische Resolution
  - Typsicherheit gegen Flexibilität

49

Wirklich nur “Syntactical Sugar”?

- Lambdas
- Functional Interfaces
- Fluent API
- Java Streams
- Parallelism, Promises, Futures, ...

51

Reflektive Programmierung

- Run-Time Type Information
  - Zur Übersetzungszeit
- Reflection API
  - Zur Laufzeit

Beispiel

- Generierung einer XML-Beschreibung für eine Java-Klasse
- Zentrale Anlaufstelle ist die Klasse **`Class`**
  - **`getDeclaredConstructors`**
    - liefert Array aller definierten Konstruktoren – unabhängig von der Sichtbarkeit
  - **`getConstructors`**
    - liefert Array der sichtbaren Konstruktoren
  - **`getDeclaredFields`** und **`getFields`**
  - **`getDeclaredMethods`** und **`getMethods`**
  - ...
- Zugriff auf ALLE sprach-relevanten Eigenschaften eines Typs

Class TalkAbout

```
package syssoft.reflection;

import java.io.*;
import java.lang.reflect.*;

public class TalkAbout {
        private Class c;

        public TalkAbout ( Class candidate ) {
                c = candidate;
        }

        public void print ( PrintStream out ) {
                out.println("<?xml version=\"1.0\" encoding=\"UTF-8\">");
                out.println("<class name=\"" + c.getCanonicalName() + "\">");
                printConstructors(c.getDeclaredConstructors(),out);
                printDeclaredFields(c.getDeclaredFields(),out);
                out.println("</class>");
        }

        private void printConstructors ( Constructor ctors[], PrintStream out ) {
                for (Constructor ctor : ctors ) {
                        out.println("\t<constructor name=\"" + ctor.getName() + "\" modifiers=\"" +
                                Modifier.toString(ctor.getModifiers()) + "\" >");
                        printArgumentTypes(ctor.getParameterTypes(),out);
                        out.println("\t</constructor>");
                }
        }

        private void printDeclaredFields ( Field fields[], PrintStream out ) {
                for (Field field : fields ) {
                        out.println("\t<field name=\"" + field.getName() + "\" type=\"" +
                                field.getType().toString() + "\" modifiers=\"" +
                                Modifier.toString(field.getModifiers()) + "\" />");
                }
        }

        private void printArgumentTypes ( Class argts[], PrintStream out ) {
                for (Class arg : argts) {
                        out.println("\t\t<type name=\"" + arg.getName() + "\" />");
                }
        }
```

Untersuchte Klasse

```
package syssoft.reflection;

public class Candidate {
    public Candidate ( int x) { this.x = x; }

    public Candidate () { Reset(); }

    private int x;

    public int get_x () { return x; }

    public void f ( int y, double d ) {
        if (d < 0) x = y; else x = -y;
    }

    protected void Reset () { x = 42; }
}
```

Ergibt:

```
<?xml version="1.0" encoding="UTF-8">
<class name="syssoft.reflection.Candidate">
     <constructor name="syssoft.reflection.Candidate" modifiers="public" >
          <type name="int" />
     </constructor>
     <constructor name="syssoft.reflection.Candidate" modifiers="public" >
     </constructor>
     <field name="x" type="int" modifiers="private" />
</class>
```

Ziele

- Anreicherung des Programmcodes
- Anwendungsspezifische Erweiterbarkeit
- Überprüfbarkeit durch Compiler
- Auswertung der Annotationen
  - Verarbeitung der Quellen
  - Übersetzung
  - Laufzeit
- Scope einer Annotation
  - Auf welche Elemente eines Typs bezieht sich die Annotation?
    - Klasse, Interface, Methode, ...
- Vergleichbar Attributen in .NET

Scope

- ElementType

Enum Constant Summary
ANNOTATION_TYPE Annotation type declaration
CONSTRUCTOR Constructor declaration
FIELD Field declaration (includes enum constants)
LOCAL_VARIABLE Local variable declaration
METHOD Method declaration
PACKAGE Package declaration
PARAMETER Parameter declaration
TYPE Class, interface (including annotation type), or enum declaration

Retention

- Reichweite der Annotation

Enum Constant Summary
CLASS Annotations are to be recorded in the class file by the compiler but need not be retained by the VM at run time.
RUNTIME Annotations are to be recorded in the class file by the compiler and retained by the VM at run time, so they may be read reflectively.
SOURCE Annotations are to be discarded by the compiler.

Beispiel

- Ausgangspunkt TalkAbout (Beispiel Reflection)
  - Methoden können über Annotation um Kommentare erweitert werden, die an entsprechender Stelle in generierten XML-Beschreibung erscheinen sollen
- Definition der Annotation **`Comment`**

```
package syssoft.annotation;

import java.lang.annotation.*;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Comment {
    String value() default "";
}
```

Verwendung in TalkAbout

- Zugang ebenfalls über Klasse **`Class`**
  - **`getAnnotation`** oder **`getAnnotations`**

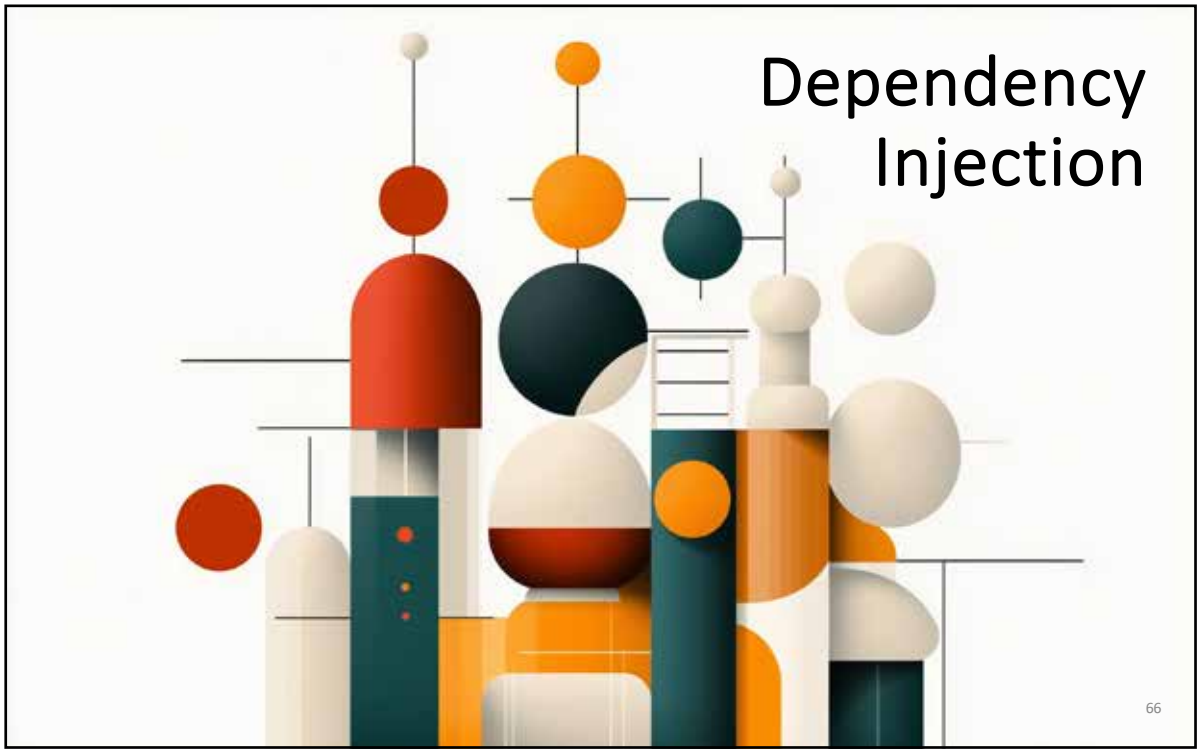
```
private void printDeclaredMethods ( Method ms[], PrintStream out ) {
     for (Method m : ms ) {
          out.println("\t<method name=\"" + m.getName() +
                 "\" modifiers=\"" + Modifier.toString(m.getModifiers()) +
                "\" return_type=\"" + m.getReturnType().getName() +
                "\">");
          printArgumentTypes(m.getParameterTypes(),out);
          Comment c = m.getAnnotation(Comment.class);
          if (c != null) {
               out.println("\t\t<comment>");
               out.println("\t\t\t" + c.value());
               out.println("\t\t</comment>");
          }
          out.println("\t</method>");
     }
}
```

Verwendung der Annotation

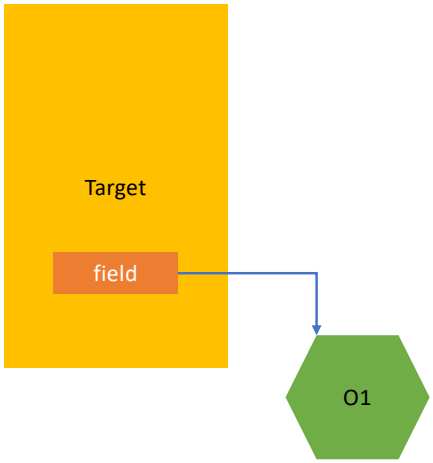
```
package syssoft.reflection;

public class Candidate {
     public Candidate ( int x) { this.x = x; }

     public Candidate () { Reset(); }

     private int x;

     @Comment("Hier kann man x lesen")
     public int get_x () { return x; }

     public void f ( int y, double d ) {
          if (d < 0) x = y; else x = -y;
     }

     protected void Reset () { x = 42; }
}
```

Und das Ergebnis

```
?xml version="1.0" encoding="UTF-8" ?>
<class name="syssoft.annotation.Candidate">
     <constructor name="syssoft.annotation.Candidate" modifiers="public" >
          <type name="int" />
     </constructor>
     <constructor name="syssoft.annotation.Candidate" modifiers="public" >
     </constructor>
     <field name="x" type="int" modifiers="private" />
     <method name="get_x" modifiers="public" return_type="int">
          <comment>
               Hier kann man x lesen
          </comment>
     </method>
     <method name="f" modifiers="public" return_type="void">
          <type name="int" />
          <type name="double" />
     </method>
     <method name="Reset" modifiers="protected" return_type="void">
     </method>
</class>
```

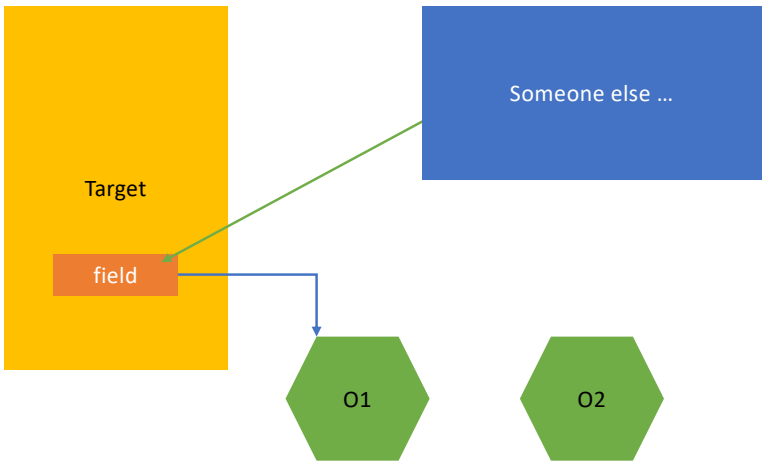


Dependency Injection



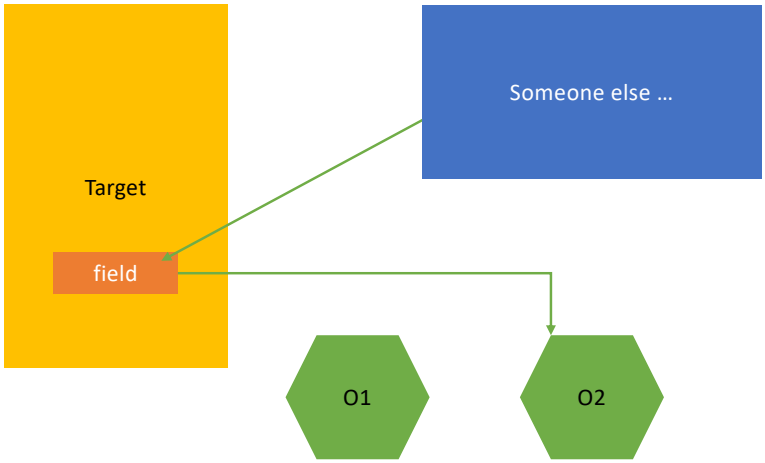
67

Dependency Injection



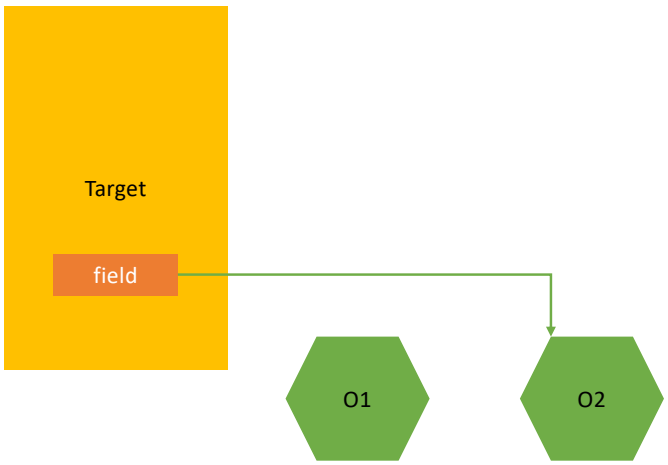
68

Dependency Injection



69

Dependency Injection



70

C / C++

71


O'REILLY
Effective Modern C++
Scott Meyers
Modern C
MEAP
O'REILLY
21st Century C
Ben Klemens

72

C++ Evolution

C++

```
// circle and shape are user-defined types
circle* p = new circle( 42 );
vector<shape*> v = load_shapes();

for( vector<circle*>::iterator i = v.begin(); i != v.end(); ++i ) {
    if( *i && **i == *p )
        cout << **i << " is a match\n";
}

for( vector<circle*>::iterator i = v.begin();
        i != v.end(); ++i ) {
    delete *i; // not exception safe
}

delete p;
```

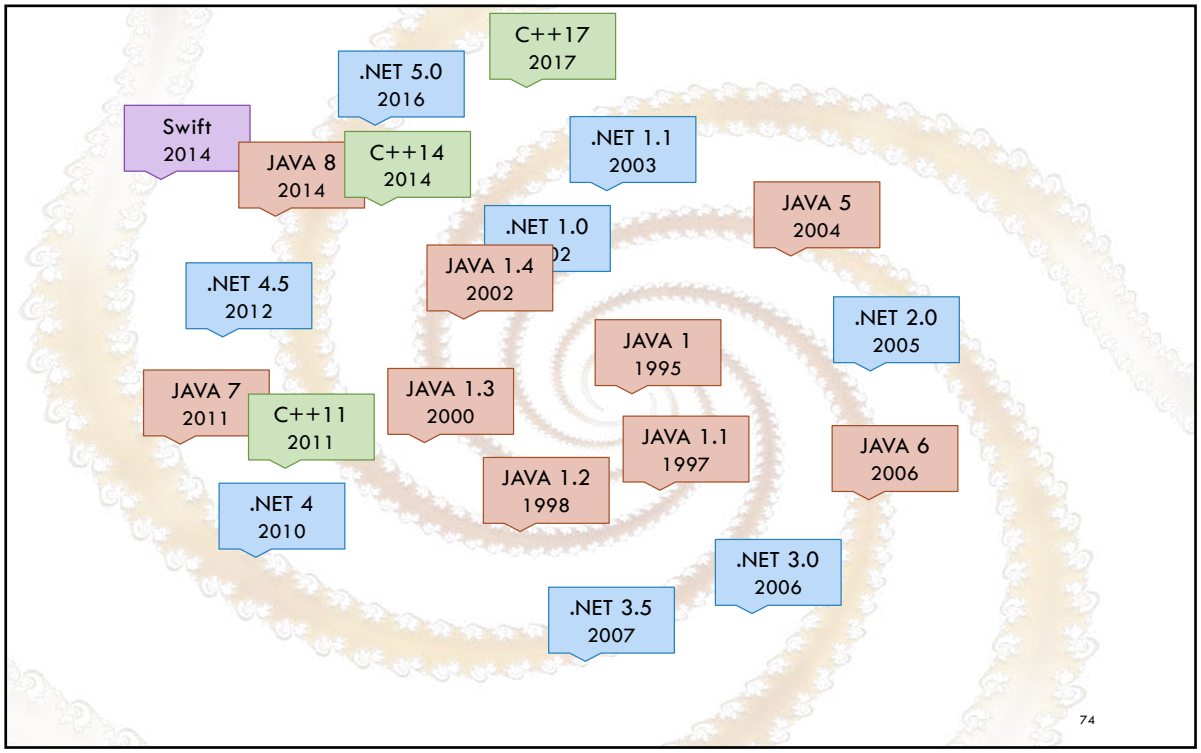
Here's how the same thing is accomplished in modern C++:

C++

```
#include <memory>
#include <vector>
// ...
// circle and shape are user-defined types
auto p = make_shared<circle>( 42 );
vector<shared_ptr<shape>> v = load_shapes();

for_each( begin(v), end(v), [&]( const shared_ptr<shape>& s ) {
    if( s && *s == *p )
        cout << *s << " is a match\n";
} );
```

http://msdn.microsoft.com/en-us/library/hh279654.aspx

73



74

Aktuelle Versionen

- Java 23 (September 2024)
- .NET 9.0 (November 2024)
- C++23

75