

Vorlesung ProInformatik III

Objektorientierte Programmierung

Freie Universität Berlin



Prof. Dr. Marco Block-Berlitz
Sommersemester 2013

Organisatorisches

Zeitlicher Ablauf

- | | |
|-------------------|--|
| 08:30 – 11:00 Uhr | 2 Vorlesungsteile mit ca. 15 Minuten Pause |
| 11:00 – 12:15 Uhr | Mittagspause |
| 12:15 – 15:45 Uhr | Theoretische und praktische Übungen
(aktive und erfolgreiche Teilnahme) |



Tutoren und Tutorien



Michael Kmoch Eduard J. Wolf Toni Schiemank Benjamin Gehmlich

Scheinkriterien

Übungszettel werden nach Bedarf am Ende einer Vorlesung ausgegeben
(60% der Aufgaben erfolgreich bearbeiten, n-1 Zettel mehr als 20%)

Klausur: **vorletzter Vorlesungstag (29.08.2013), 8:30 Uhr s.t., 90 Minuten**
(Bestehen der Klausur)



Vorlesungsmaterialien und Literaturliste

Vorlesungsmaterialien

Vorlesungsfolien werden bereit gestellt und ersetzen das Vorlesungsskript. Programme, Literaturhinweise und weiteres Material werden auf einer Veranstaltungswebseite zur Verfügung gestellt:

http://page.mi.fu-berlin.de/block/pi3_2013.html



Literaturliste

- [1] Hoffmann D.W.: "*Theoretische Informatik*", Hanser-Verlag 2009
- [2] Kleuker S.: "*Formale Modelle der Softwareentwicklung - Model-Checking, Verifikation, Analyse und Simulation*", Vieweg-Teubner-Verlag 2009
- [3] Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*", 2. Auflage, Springer-Verlag 2009
- [4] Gries D., Gries P.: "*Multimedia Introduction to Programming Using Java*", Springer-Verlag 2005
- [5] Weiss M.A.: "*Data Structures & Problem Solving Using Java*", 3. Auflage, Addison-Wesley-Verlag 2005
- [6] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.: "*Introduction to Algorithms*", 3. Auflage, MIT Press 2009
- [7] Martin R.C.: „*Clean Code*“, Prentice Hall International Verlag 2008



Diskussionsplattform - Forum

Mailingliste:

werden wir nicht brauchen - wichtige Änderungen und Mitteilungen erscheinen im Forum

Forum:

<http://www.java-uni.de>

Es gibt einen passwortgeschützten Bereich nur für Euch:

FU SoSe 2013: Proinformatik - Algorithmen und Programmierung II
("proinf")

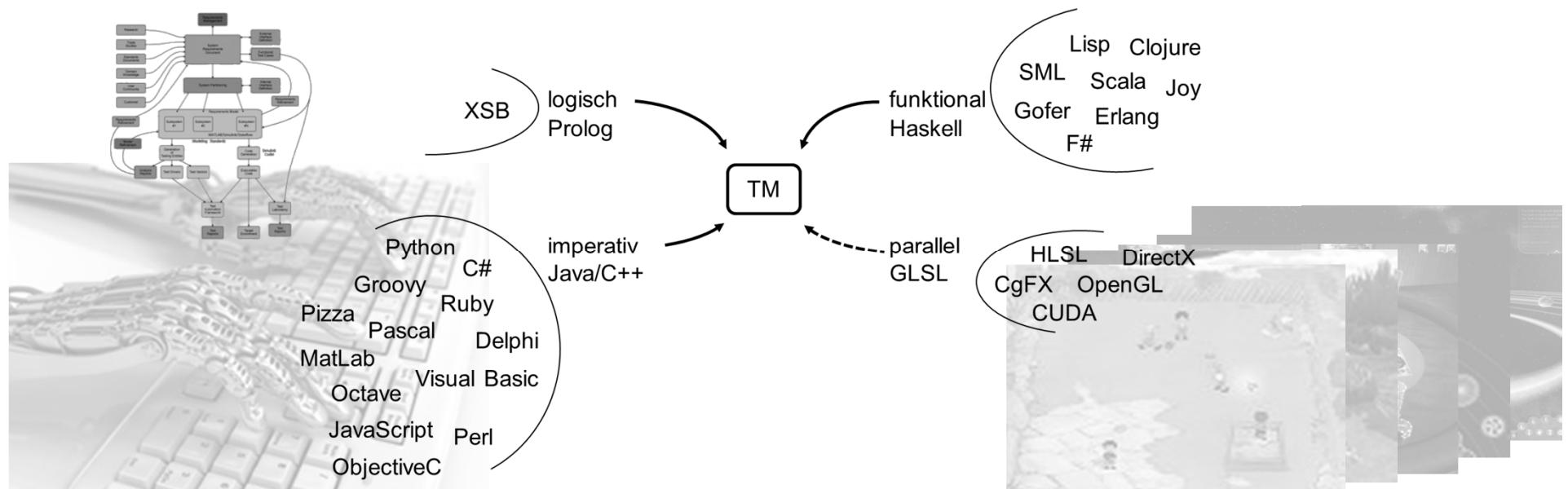
Das Java-Forum dient als Diskussionsplattform.

Helft Euch gegenseitig!



Vorlesungsteil

Motivation und Einführung



Chef ist nicht der, der etwas tut, sondern der das Verlangen weckt, etwas zu tun.
Edgar Pisani



Wer von Euch kann Programmieren?

||||| |||||

||||| |||||

Unsere wichtigsten Themen und Ziele

Grundlagen der Berechenbarkeit/Programmiermethodik

- Syntax und operationelle Semantik imperativer Programmiersprachen
- Einführung und Ausbildung zum Selbststudium
- Lernen objektorientiert zu denken und zu programmieren

Einführung in die Laufzeit- und Komplexitätsanalyse

- Landau-Symbolik
- Kleine Programmabschnitte analysieren und vergleichen können
- Effiziente Lösungen für neuartige Probleme formulieren

Formale Verfahren zur Spezifikation und Verifikation imperativer Programme bzw. Testtechniken

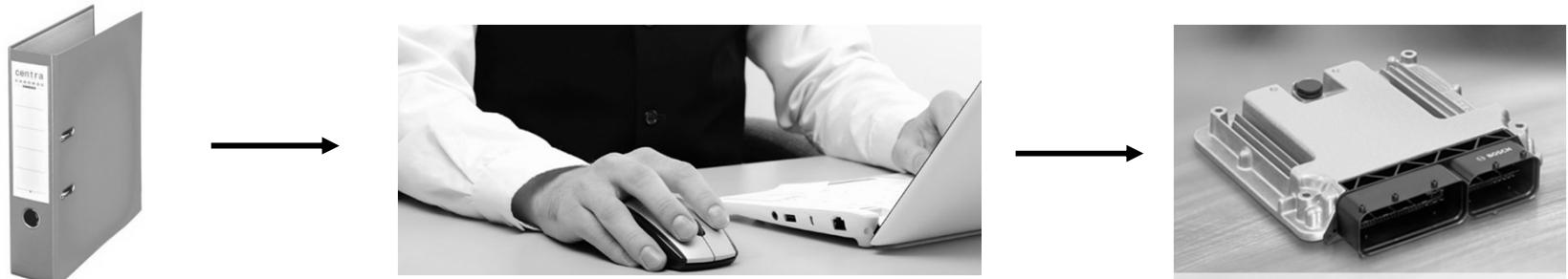
- Programme möglichst fehlerfrei schreiben
- Entworfene Programme testen
- Test-Driven-Development beherrschen

Wie wähle ich die passende Programmiersprache
für ein Projekt?

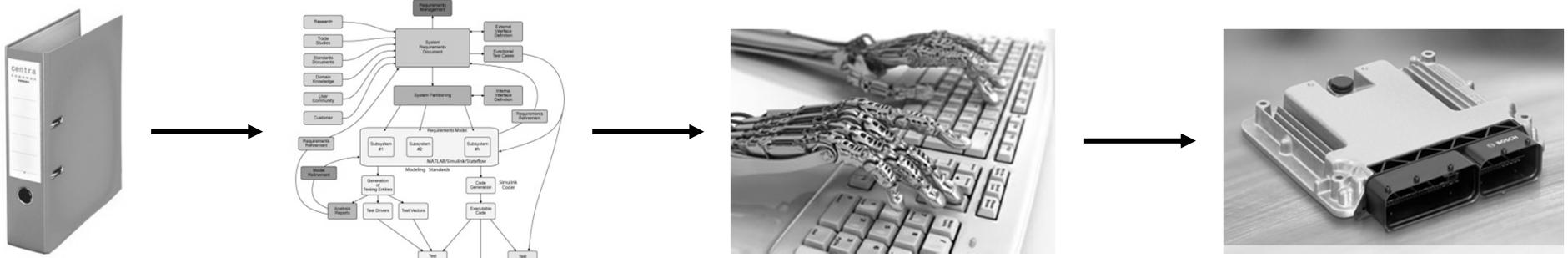
Automobilindustrie



Klassische Softwareentwicklung



Modell-basierte Softwareentwicklung (seit 2006)



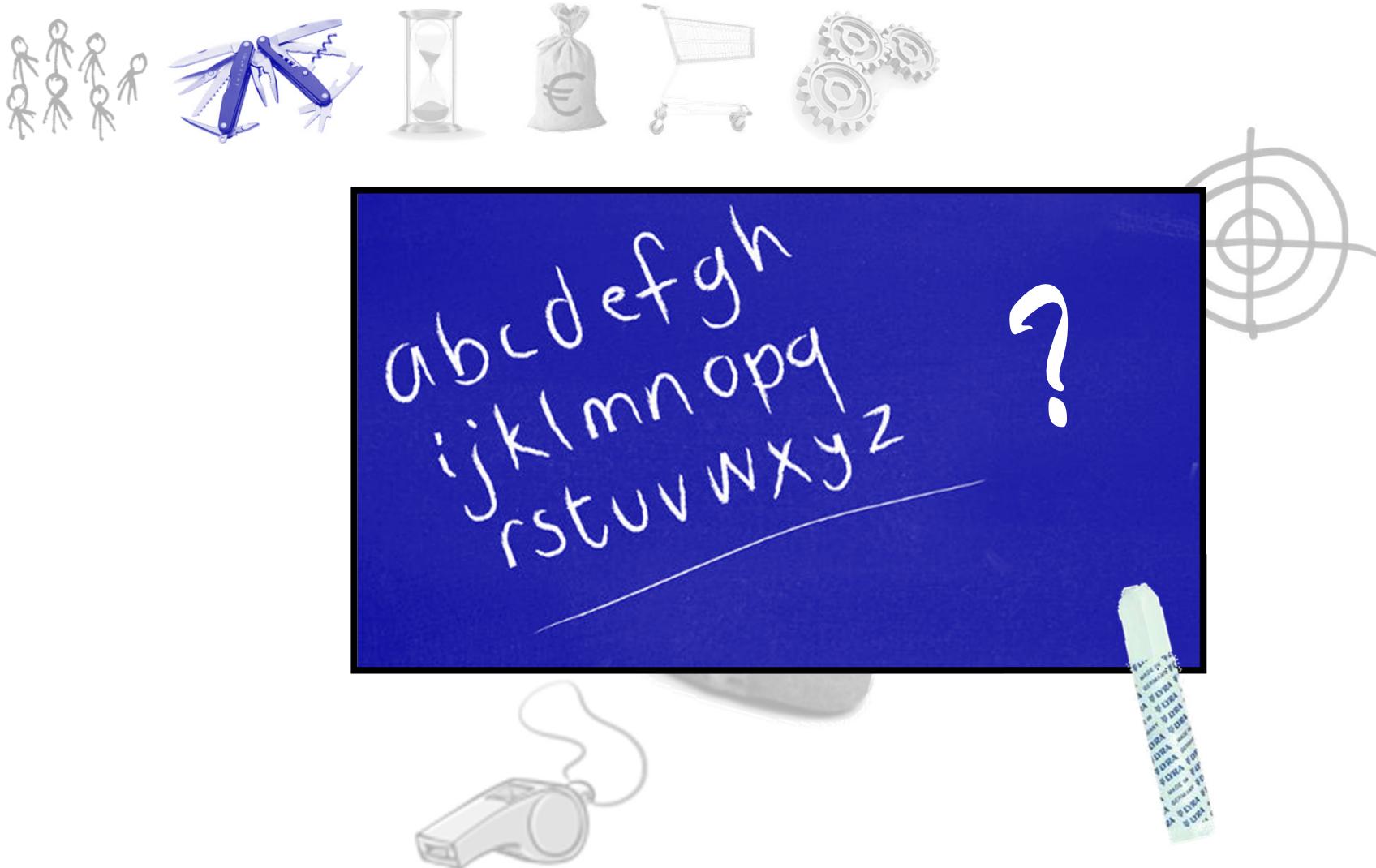
Softwareentwicklungsprozess



Softwareentwicklungsprozess



Softwareentwicklungsprozess



Motivation?

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VO
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	VFs
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothon	Slate	Visual Prolog
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Pacbase	Profan	Sleep	Web
Amber Smalltalk	C	CSilber	FoxPro	jProfan	MDL	Paradox	Puck	Smalltalk	Whitespace
AML	C++	CURL	Fpii	JScript	Mercury	Pascal	PureBasic	Smartware	Winbatch
AMOS	C#	Curry	FPr	JSP	Mesa	Pawn	Pure Data	SNOBOL4	WMLScript
AMPL	C/AL	D	FreeBASIC	JustBASIC	Miranda	PEARL	Python	Sonnyscript	Wlanguage
Angoss	Call Proc.Lang.	DarkBASIC	G	Jython	ML	Perl	Q	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Phalanger	R	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

Motivation!

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VQ
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	WEB
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothom	Visual Prolog	
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Paradise	Prfml	Sleep	Web
Amber	Smalltalk	C	CSilber	FoxPro	iProf	NDL	Paradox	Smalltalk	Whitespace
AML		C++	CUP!	GWT	JavaScript	Mercury	Pascal	Smartware	Winbatch
AMOS		C#	Cutty	HP	JSP	Mesa	Pawn	SNOBOL4	WMLScript
AMP		CAL	D	FreeBASIC	JustBASIC	Miranda	PEARL	Sonnyscript	Wlanguage
Angoss	Call Proc.Lang.	DarkBASIC	G	Jython	ML	Perl	Python	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Phalanger	Q	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

Von Programmiersprachen auf Paradigmen abstrahieren

Wichtiger Verständnisbaustein

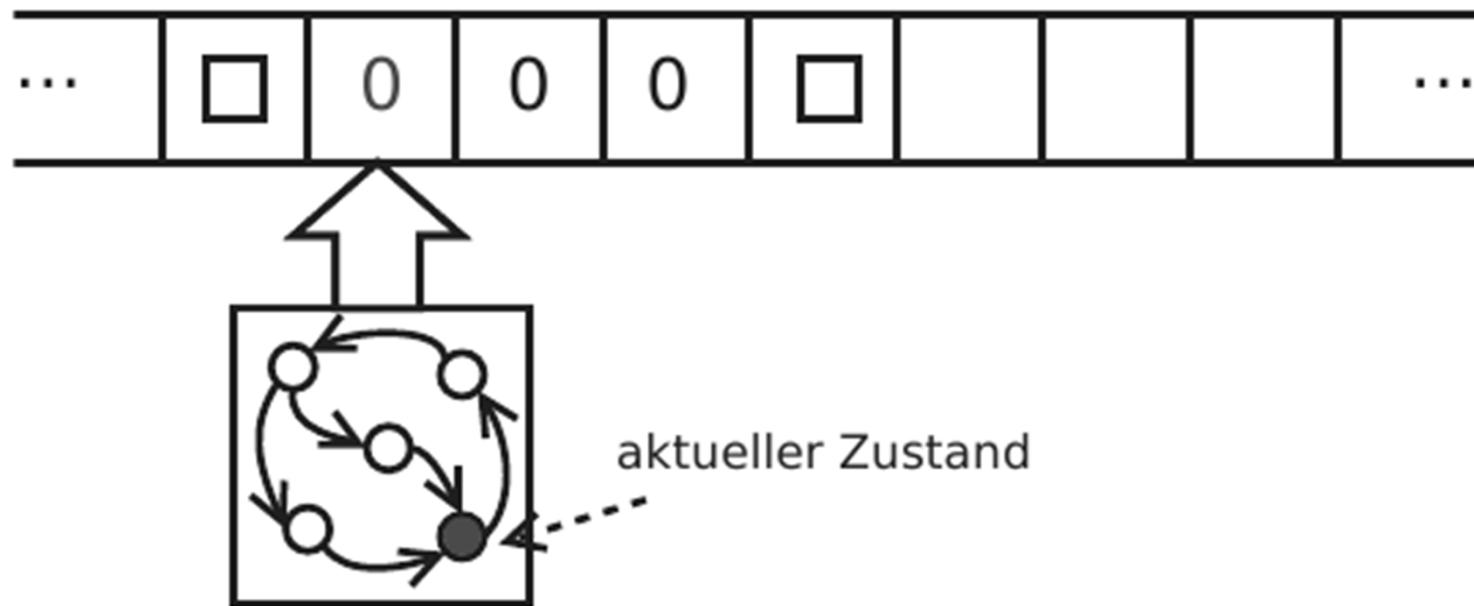


1936: Lambda-Kalkül (Church/Kleene)

$$\begin{aligned} (\times) \ 2 \ 3 &\equiv (\lambda mnf.m(nf))(\lambda sz.s(sz))(\lambda sz.s(s(sz))) \\ &\stackrel{\beta}{\equiv} (\lambda nf.(\lambda sz.s(sz))(nf))(\lambda sz.s(s(sz))) \\ &\stackrel{\beta}{\equiv} \lambda f.(\lambda sz.s(sz))((\lambda sz.s(s(sz)))f) \\ &\stackrel{\beta}{\equiv} \lambda fz.((\lambda sz.s(s(sz)))f)((\lambda sz.s(s(sz)))f)z \\ &\stackrel{\beta}{\equiv} \lambda fz.(\lambda z.(f(f(fz))))((\lambda sz.s(s(sz)))f)z \\ &\stackrel{\beta}{\equiv} \lambda fz.f(f(f(((\lambda s.(\lambda z.s(s(sz))))f)z))) \\ &\stackrel{\beta}{\equiv} \lambda fz.f(f(f((\lambda z.f(f(fz))))z))) \\ &\stackrel{\beta}{\equiv} \lambda fz.f(f(f(f(f(fz)))))) \end{aligned}$$

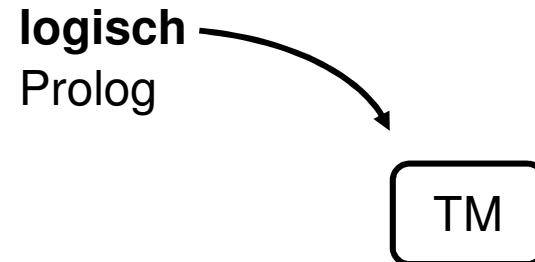
Berechenbarkeit = Lambda-Kalkül

1936: Turing-Maschine



Berechenbarkeit = Turing-Maschine

Programmierparadigmen klassisch

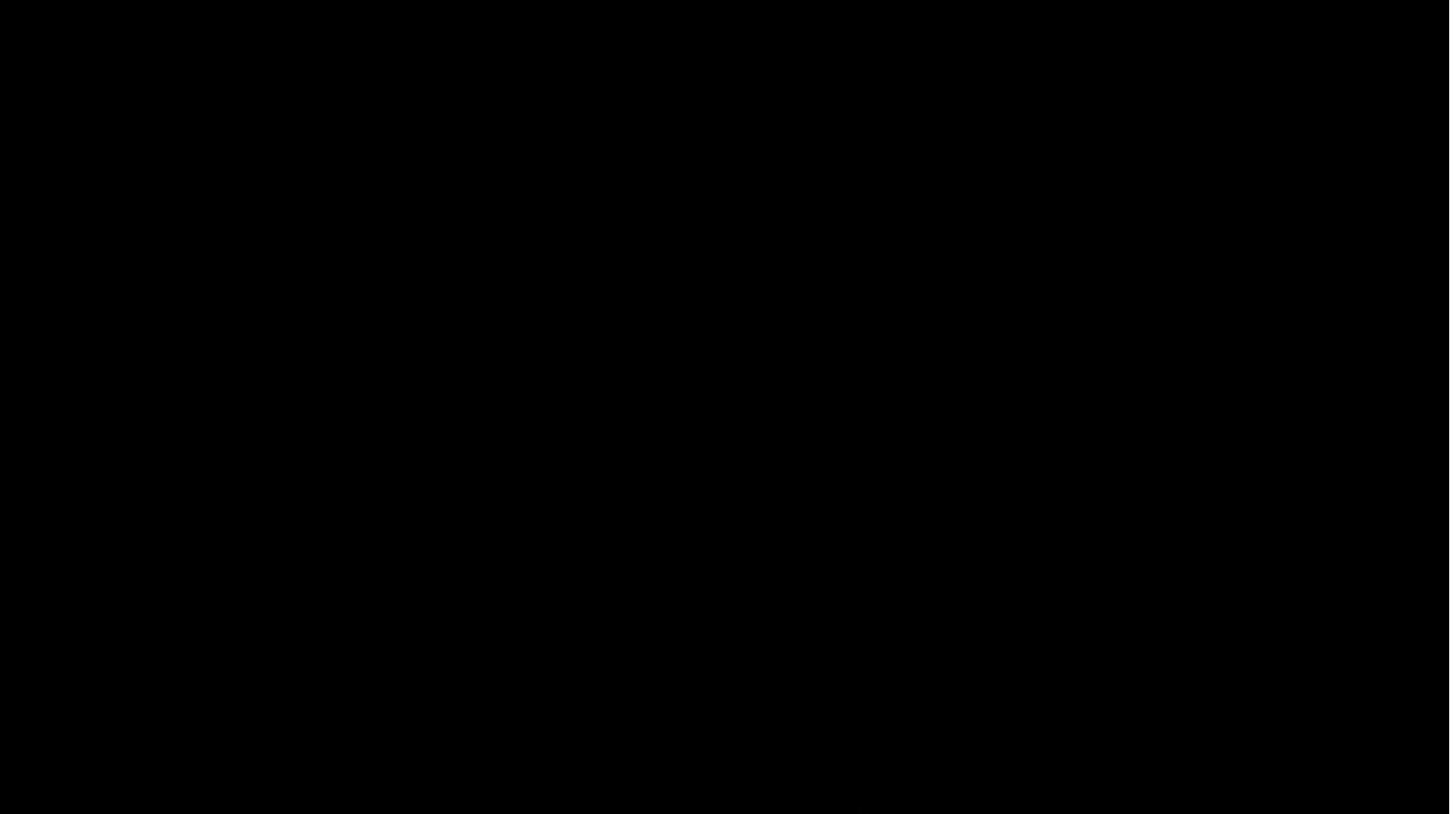


Element in eine Liste einfügen:

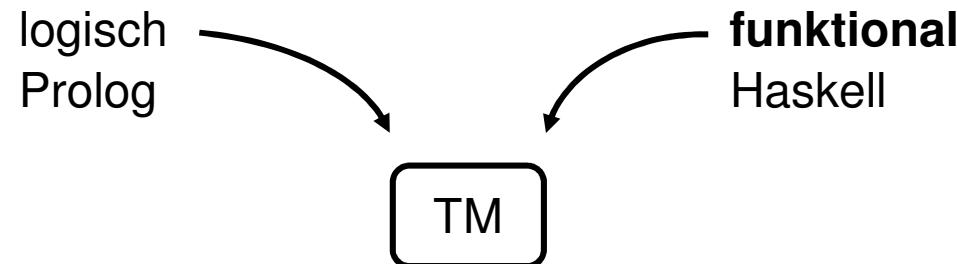
```
insert (X, List, List_with_X) :- delete (X, List_with_X, List).
```

Logische Programmierung

IBM and the Jeopardy Challenge (2010)



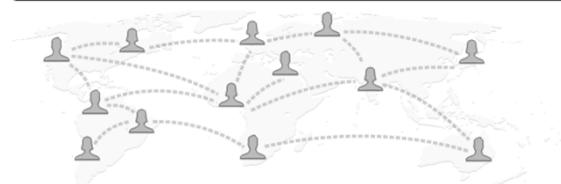
Programmierparadigmen klassisch



QuickSort-Algorithmus:

```
qSort []      = []
qSort (x:xs)  = qSort [n | n <- xs, n < x] ++ [x] ++ qSort [n | n <- xs, n >= x]
```

Funktionale Programmierung



ejabberd (XMPP-Server)

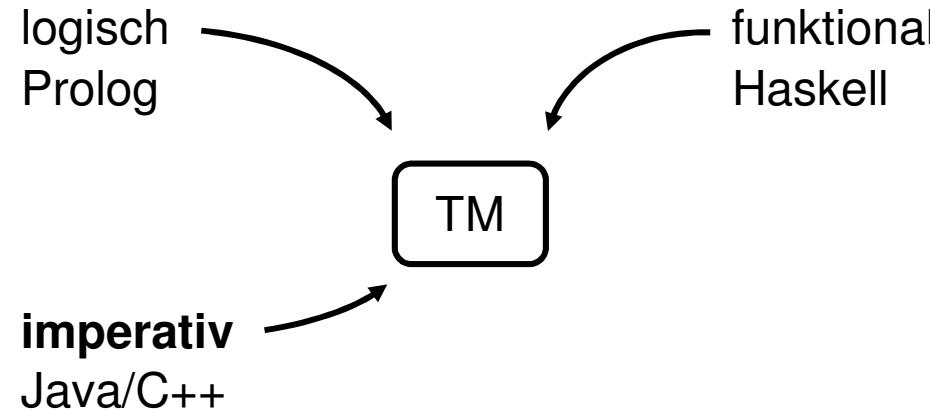


Erlang



Frag - Egosshooter in Haskell

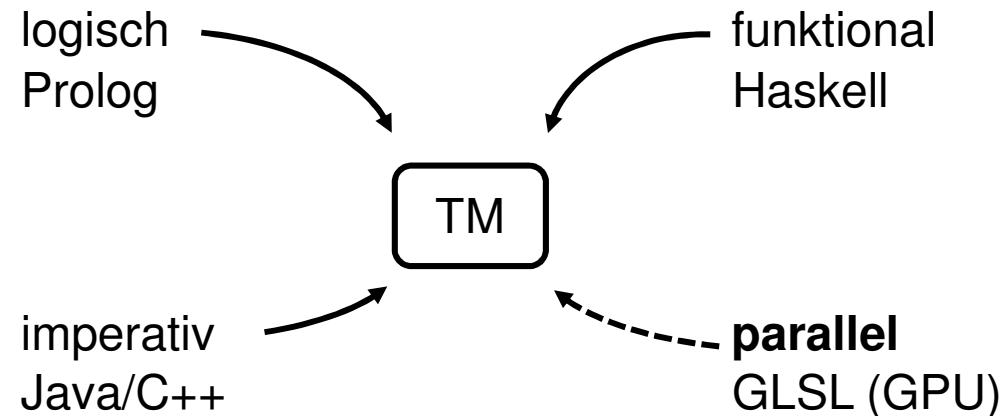
Programmierparadigmen klassisch



QuickSort-Algorithmus:

```
public void quicksort(int lo, int hi) {  
    int i=lo, j=hi;  
    int x=a[(lo+hi)/2];  
  
    while (i<=j) {  
        while (a[i]<x) i++;  
        while (a[j]>x) j--;  
        if (i<=j) {  
            int t = a[i];  
            a[i] = a[j];  
            a[j] = t;  
            i++; j--;  
        }  
    }  
    if (lo<j) quicksort(lo, j);  
    if (i<hi) quicksort(i, hi);  
}
```

Programmierparadigmen klassisch

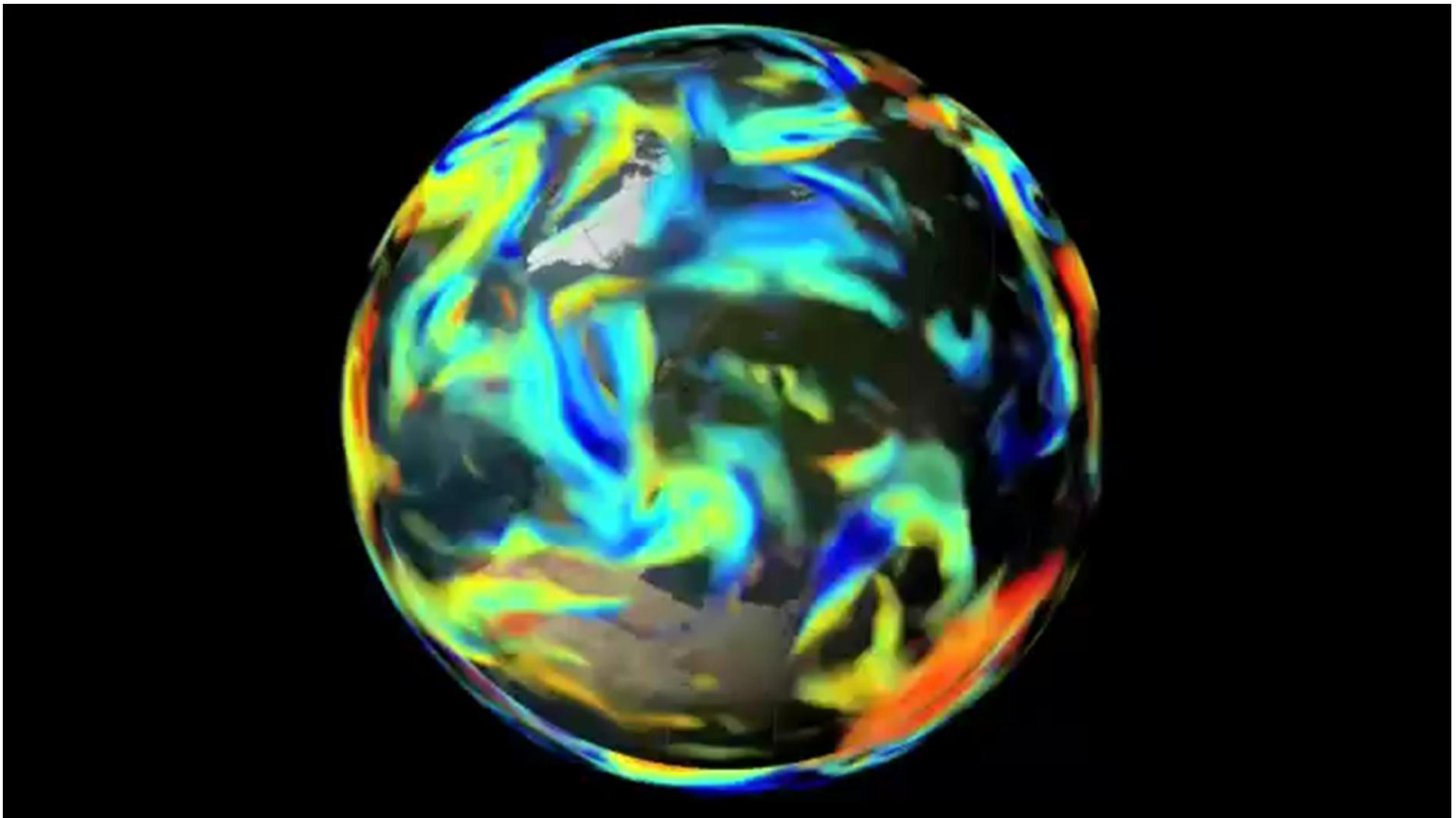


Lineare Algebra:

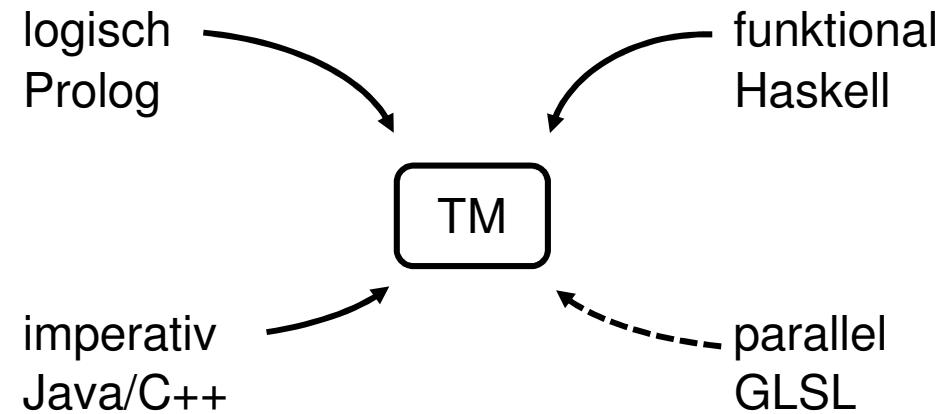
```
vec3 v3          = v1 + v2;  
vec4 coord_xyzw = vec4(v3, 1.0f);
```

Parallele Programmierung

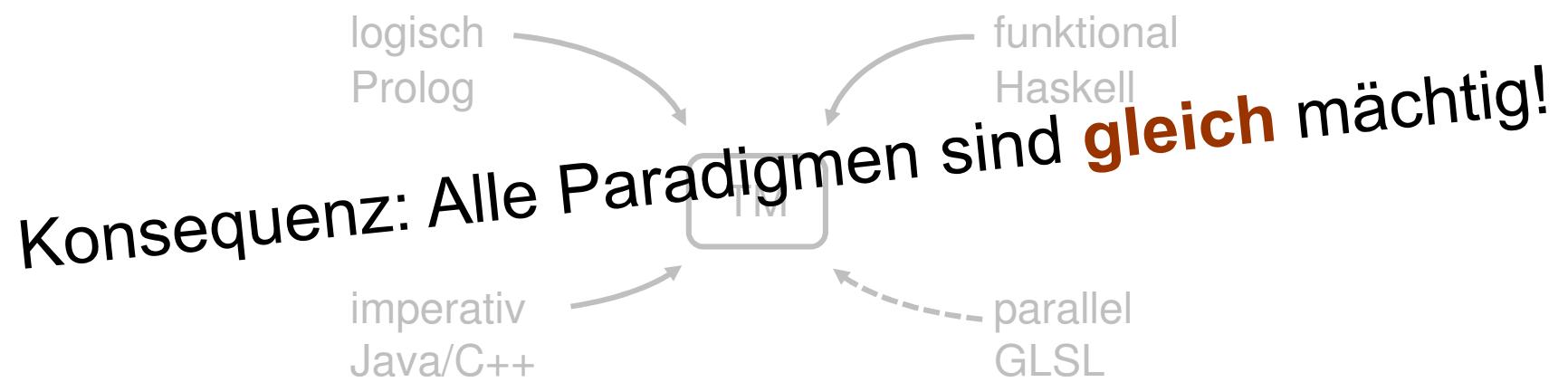
Supercomputer maps Hurricane Katrina



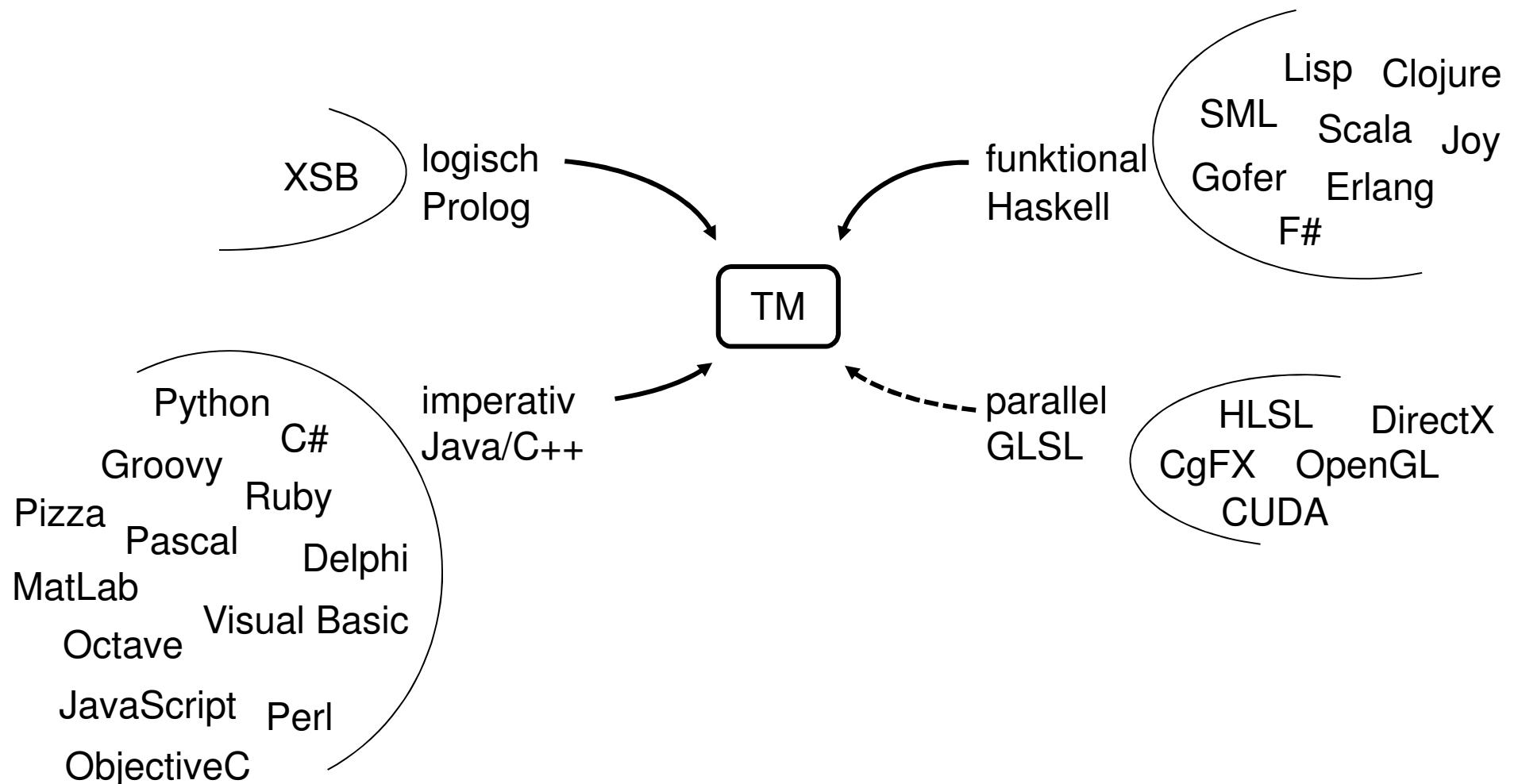
Programmierparadigmen klassisch



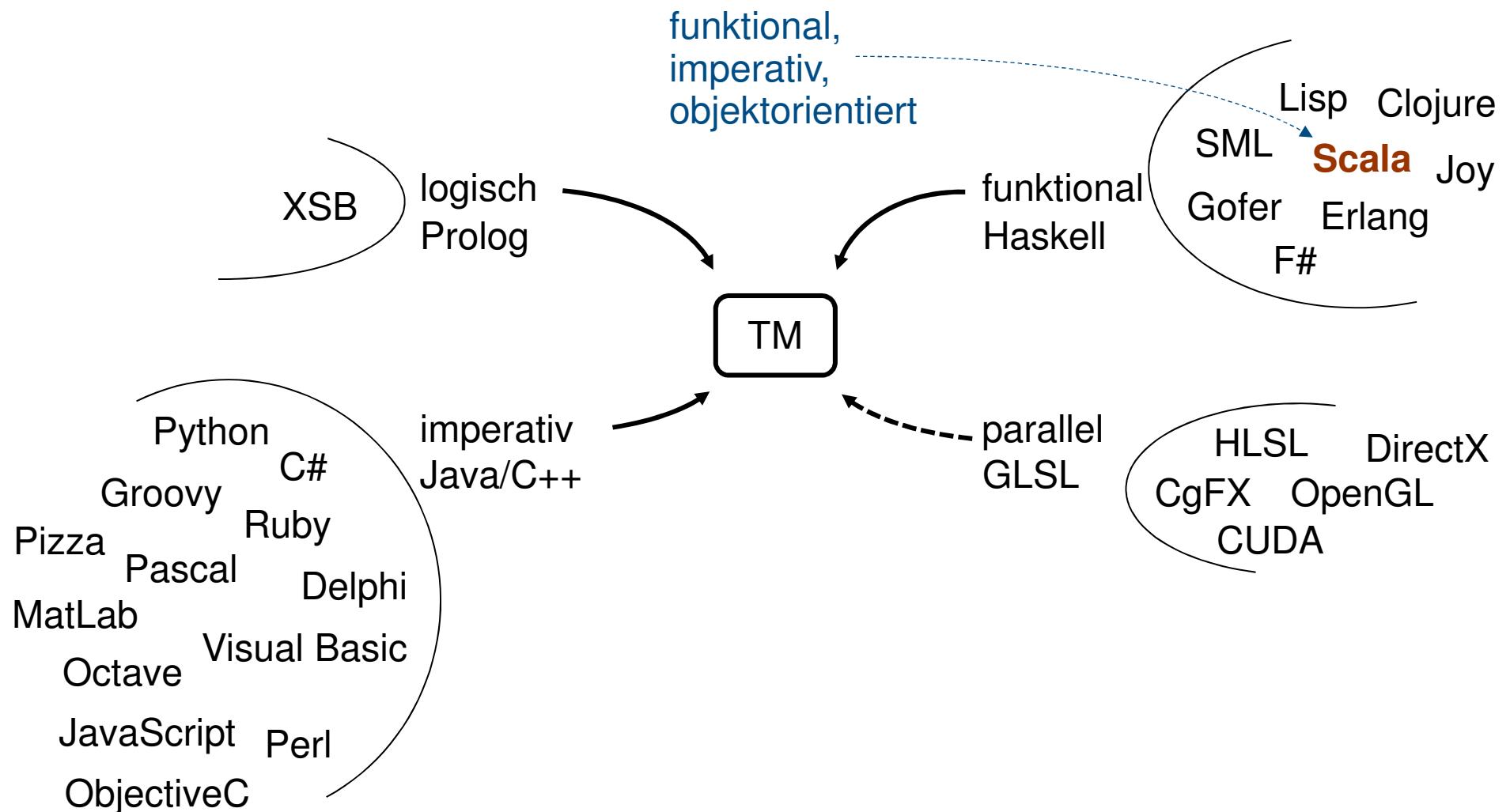
Programmierparadigmen klassisch



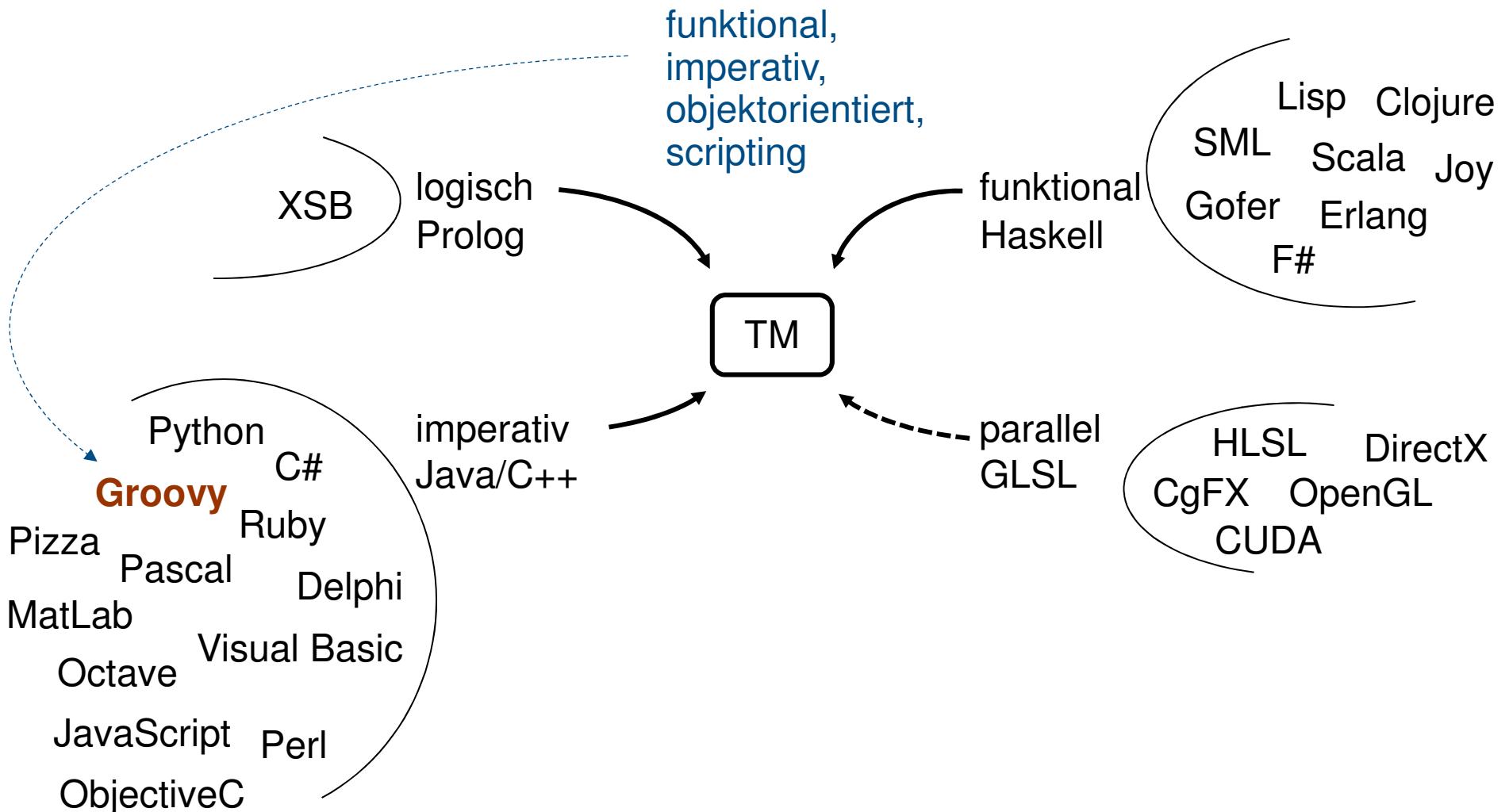
Programmierparadigmen klassisch



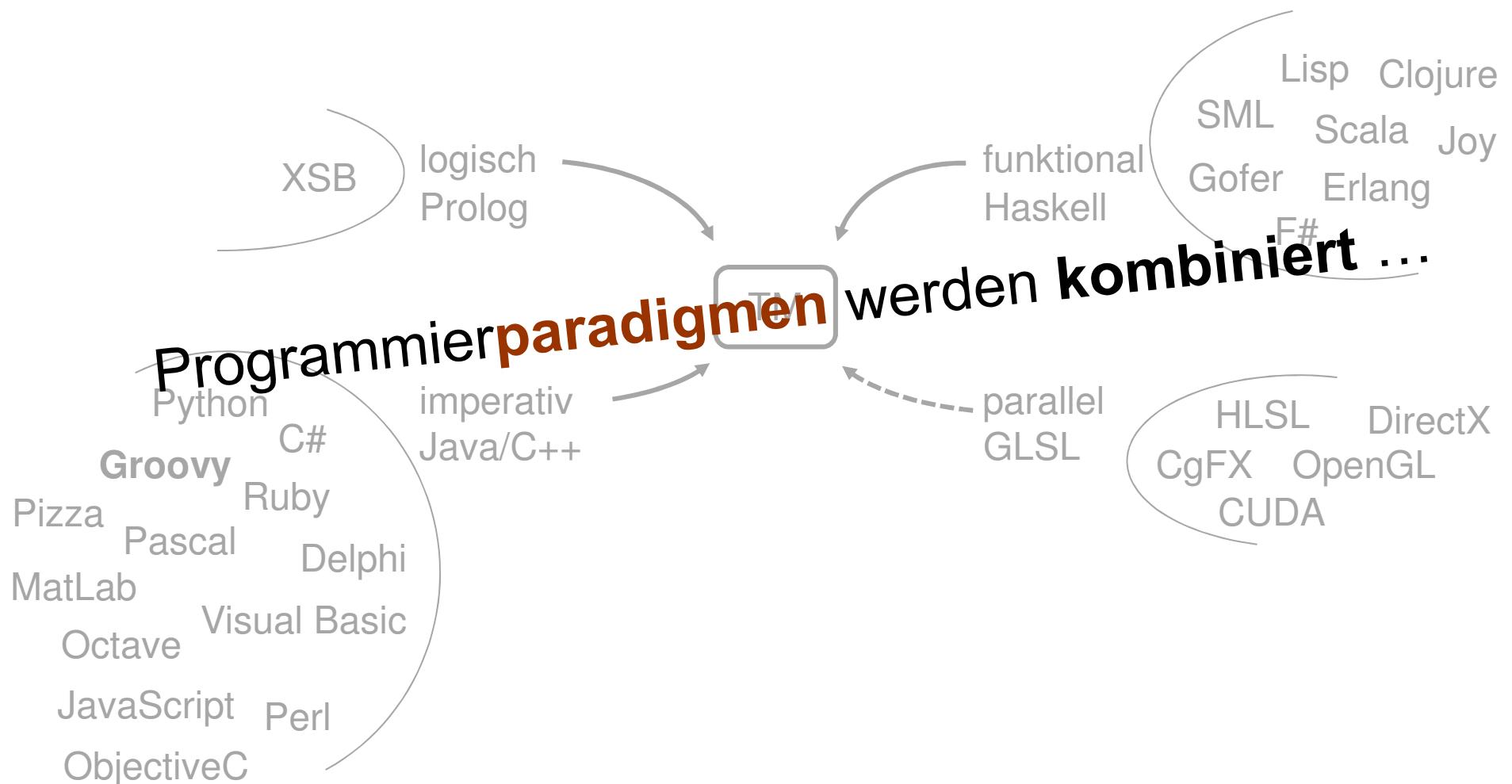
Programmierparadigmen klassisch



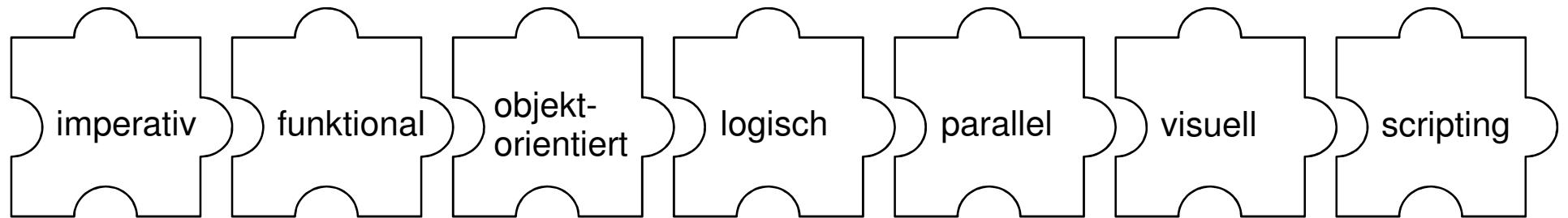
Programmierparadigmen klassisch



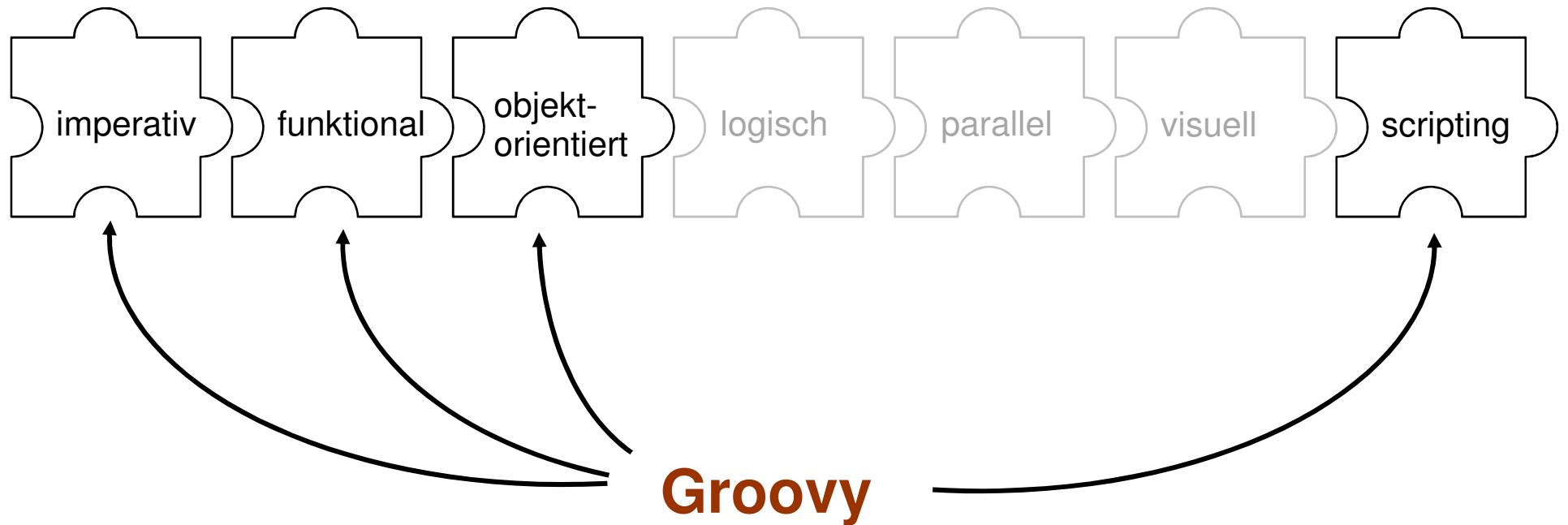
Programmierparadigmen klassisch



Programmierparadigmen modern



Programmierparadigmen modern



Einfache Syntax für Listen:

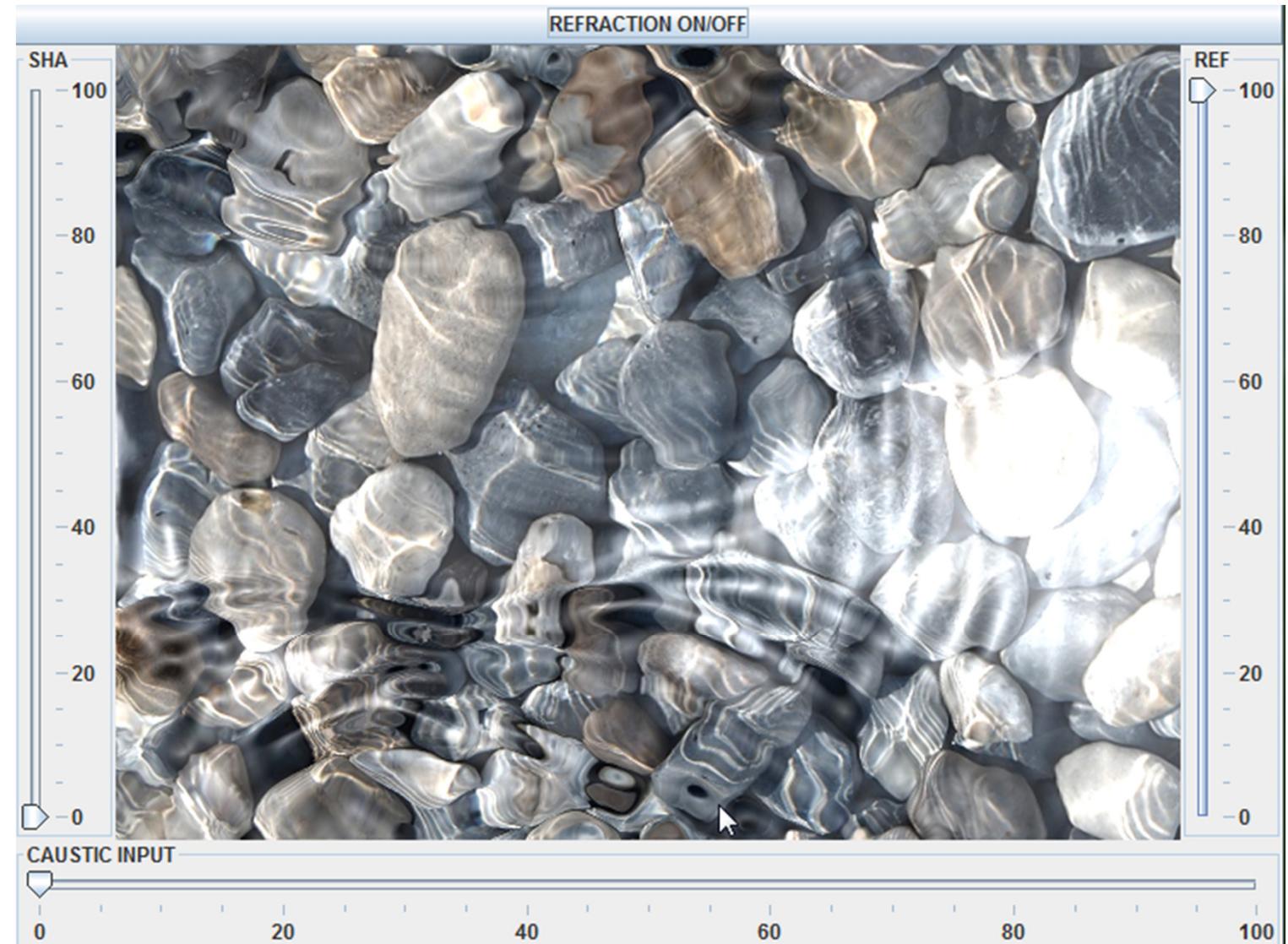
```
[ "Rod", "Carlos", "Chris" ].each{println it}
```

Integration und Kombination von Sprachen



Integration und Kombination von Sprachen

Projekt: Interaktives Wasser (2011)



Integration und Kombination von Sprachen

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VO
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	VFs
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothon	Slate	Visual Prolog
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Pacbase	Profan	Sleep	Web
Amber Smalltalk	C	CSilber	FoxPro	jProfan	MDL	Paradox	Puck	Smalltalk	Whitespace
AML	C++	CURL	Fpii	JScript	Mercury	Pascal	PureBasic	Smartware	Winbatch
AMOS	C#	Curry	FPr	JSP	Mesa	Pawn	Pure Data	SNOBOL4	WMLScript
AMPL	C/AL	D	FreeBASIC	JustBASIC	Miranda	PEARL	Python	Sonnyscript	Wlanguage
Angoss	Call Proc.Lang.	DarkBASIC	G	Jython	ML	Perl	Q	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Phalanger	R	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

Integration und Kombination von Sprachen

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VO
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	VFs
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothon	Slate	Visual Prolog
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Pacbase	Profan	Sleep	Web
Amber Smalltalk	C	CSilber	FoxPro	jProfan	MDL	Paradox	Puck	Smalltalk	Whitespace
AML	C++	CURL	Fpii	JScript	Mercury	Pascal	PureBasic	Smartware	Winbatch
AMOS	C#	Curry	FPr	JSP	Mesa	Pawn	Pure Data	SNOBOL4	WMLScript
AMPL	C/AL	D	FreeBASIC	JustBASIC	Miranda	PEARL	Python	Sonnyscript	Wlanguage
Angoss	Call Proc.Lang.	DarkBASIC	G	Jython	ML	Perl	Q	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Bluelanger	R	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

GLSL

LWJGL

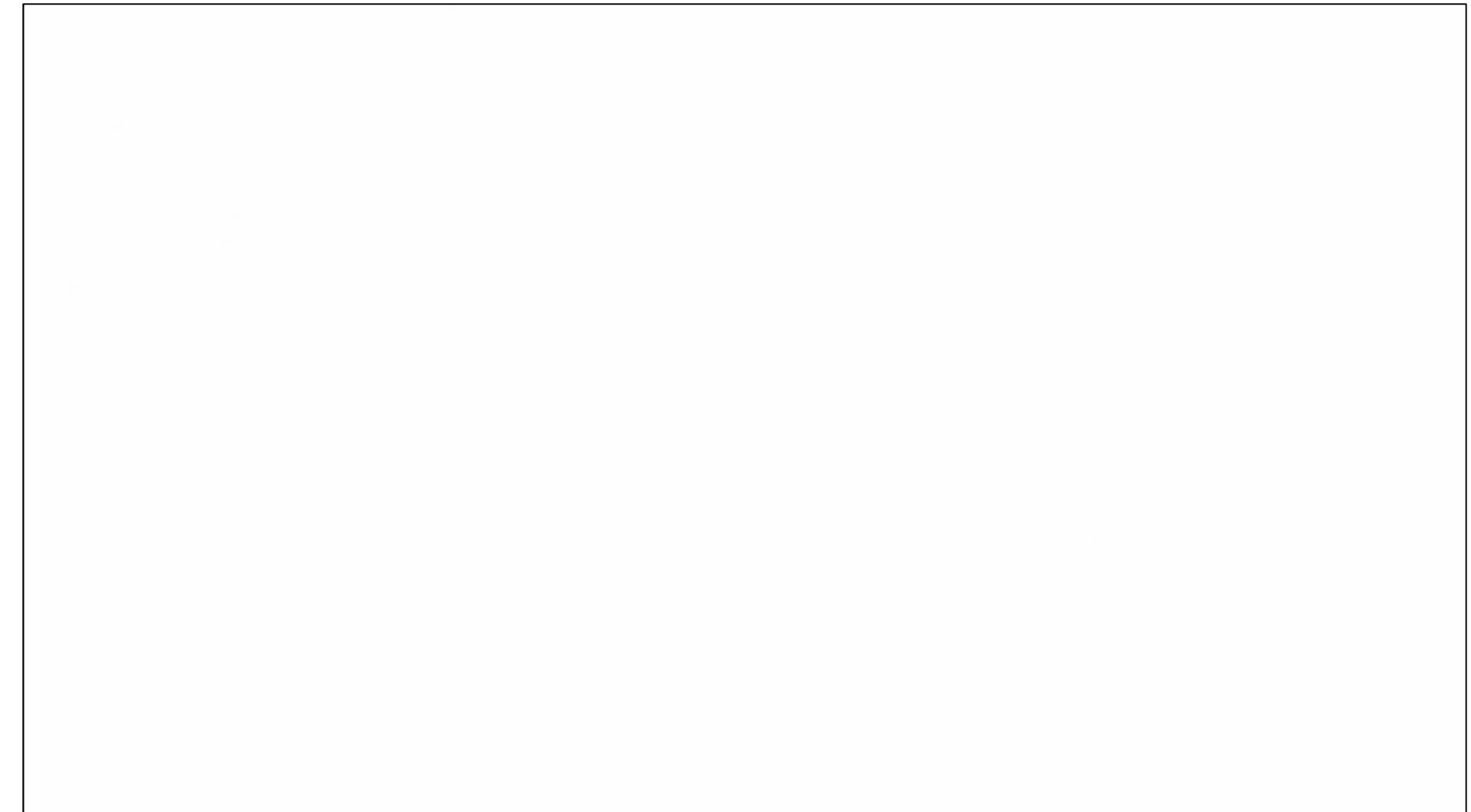
Integration und Kombination von Sprachen

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VO
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	VF
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothon	Slate	Visual Prolog
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Pacbase	Profan	Smalltalk	Web
Amber Smalltalk	C	CSilber	FoxPro	jProfan	MDL	Paganix	Space	Smalltalk	Whitespace
AML	C++	CURL	Fpii	JSocial	MDL	Pascal	PureBasic	Smartware	Winbatch
AMOS	C#	Curry	FP	SP	MATL	Mesa	Pawn	Pure Data	WMLScript
AMPL	DA	FreeBASIC	JustBASIC	JustBASIC	Miranda	PEARL	Python	SNOBOL4	Wlanguage
Angoss	CaP ProLang	DarkBASIC	G	Jython	ML	Perl	Q	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Phalanger	R	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

Programmiersprachen werden kombiniert ...

Betriebssysteme kombinieren

Projekt: Autonomes Fahrzeug (2011)



Betriebssysteme kombinieren

Projekt: Archäocopter (2013)

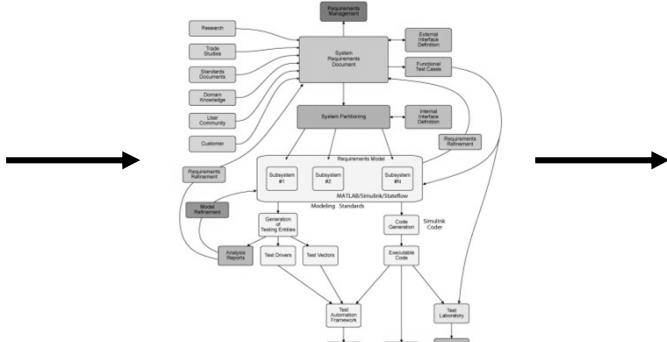


Integration und Kombination von Sprachen

A#	B	Clojure	Erlang	IBAL	LISP	O#	PL/R	SCL	Turbo_Pascal
A+	B-0	CLU	Esterel	ICI	Logo	Oberon	PL/SQL	SCPI	Tipi
A-0 - A-3	BASCOM	Cluster	Euler	Icon	Logtalk	Object REXX	PLACA	Scratch	Transact SQL
ABAP	BASIC	COBOL	Euphoria	iCon-L	LotusScript	Objective-C	Plankalkül	Scriptol	TSL
ABC	bash	Cobra	EXEC, EXEC 2	IDL	LPC	Objective-C++	Pocol	Sculptor 4GL	Turing
ActionScript	BCPL	Comal	F	Intercal	Lua	OCaml	PostScript	SDL	Typoscript
Ada	BeanShell	Comega	F#	Io	Lush	Object-Pascal	PowerBASIC	Seed7	Uniface
ADbasic	Beatnik	COMIT	Factor	Ioke	Lustre	Occam	PovRay	Self	Vala
Agda	Befunge	Common Lisp	Falcon	ISWIM	Lite-C	Octave	Processing	Sevag	VEE
AgentSpeak	BETA	Comp. Pascal	Fantom	J	M	Opa	Progres	Shakespeare	Visual Basic
Agilent VEE	BLISS	ConGolog	Faust	Jabaco	M4	Opal	Progress-4GL	Shell	VBA
AHDL	Blitz Basic	CONZEPT 16	FLOW-MATIC	Jasmin	Malbolge	OPL	Prolog	Sieve	VBScript
Aldor	BLOG	Cool	Flowcode	Java	Mantra	Open Script	Promela	Simula	VO
Aleph	Boo	COOL:GEN	Forth	JavaScript	Mathematica	Ook!	Prosa	SIRON	VF
Alice	Brainfuck	Corn	FORTRAN	JOVIAL	MATLAB	Oz	Prothon	Slate	Visual Prolog
ALGOL	Brainfuck2D	CPL	Fortress	Joy	Maxima	Pacbase	Profan	Deep	Web
Amber Smalltalk	C	CSilber	FoxPro	jProfan	MDL	Padl	PK	Smalltalk	Whitespace
AML	C++	CURL	Fpii	JSscript	Mercury	Pascal	PureBasic	Smartware	Winbatch
AMOS	C#	Curry	FP	JSF	Mesa	Pawn	Pure Data	SNOBOL4	WMLScript
AMPL	C/AL	FreeBASIC	FreeBASIC	JustBASIC	Miranda	PEARL	Python	Sonnyscript	Wlanguage
Angoss	Call ProcLang.	DarkBASIC	G	Jython	ML	Perl	Q	SP/L	X10
APL	Caml	Dart	Gambas	J#	Modula	Phalanger	R	SQL	X++
AppleScript	Cayenne	Datalog	GAP	J++	MPD	PHP	RapidBATCH	SR	XBase
AspectJ	Cecil	Datastage	GFA-BASIC	K	Mumps	Piet	REALbasic	STOS_BASIC	Xbase++
Assembler	CFML	Deadalus	Genie	Kaya	MYCIN	Pike	REBOL	Suneido	Visual XBase++
Autocoder	C for graphics	Delphi	Giotto	KIX32	MSL	PILOT	REFAL	SML	XL
Autohotkey	Chapel	DMDScript	GML	Kotlin	MPL	Pizza	REXX	StarOffice Basic	XOTcl
Autolt	Charity	Draco	Go	LabVIEW	NATURAL	PL/0	RPG	Superx++	XProfan
Avenue	Chef	DTGolog	Gofer	LALO	NetLogo	PL/B	RSL	SWiSHscript	XSLT
awk	CHILL	Dylan	GRASS	Lasso	NewtonScript	PL/I	Ruby	SML	Zer0 Tolerance
Gawk	CIP-LS	E	Groovy	Liberty Basic	NewLisp	PL/Java	Rust	TAL	Zombie
Mawk	CL	EASY	HAL	Lingo	Nemerle	PL/M	S	Tcl	Zonnon
nawk	Clarion	Eden	Haskell	Limbo	Nice	PL/P	SAIL	TECO	3APL
AWL	Clean	Eiffel	Hope	Linda	NQC	PL/Perl	Sather	TELON	3Code
AXLE	Clipper	ELAN	HQ9+	LPL	NXC	PL/pgSQL	Scala	TeX	4GL (Informix)
AXIS	CLIPS	ELF	Hypertalk	LIS	Nyquist	PL/Python	Scheme	TI_Basic	4th Dimension

Betriebssysteme werden kombiniert ..

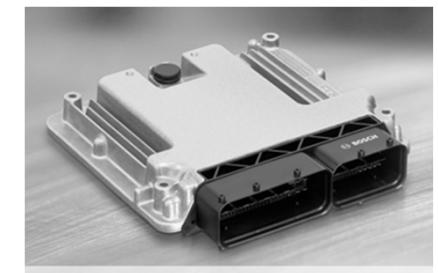
Phasen der Softwareentwicklung: Automobilindustrie



Visueller, abstrakter
Prototyp



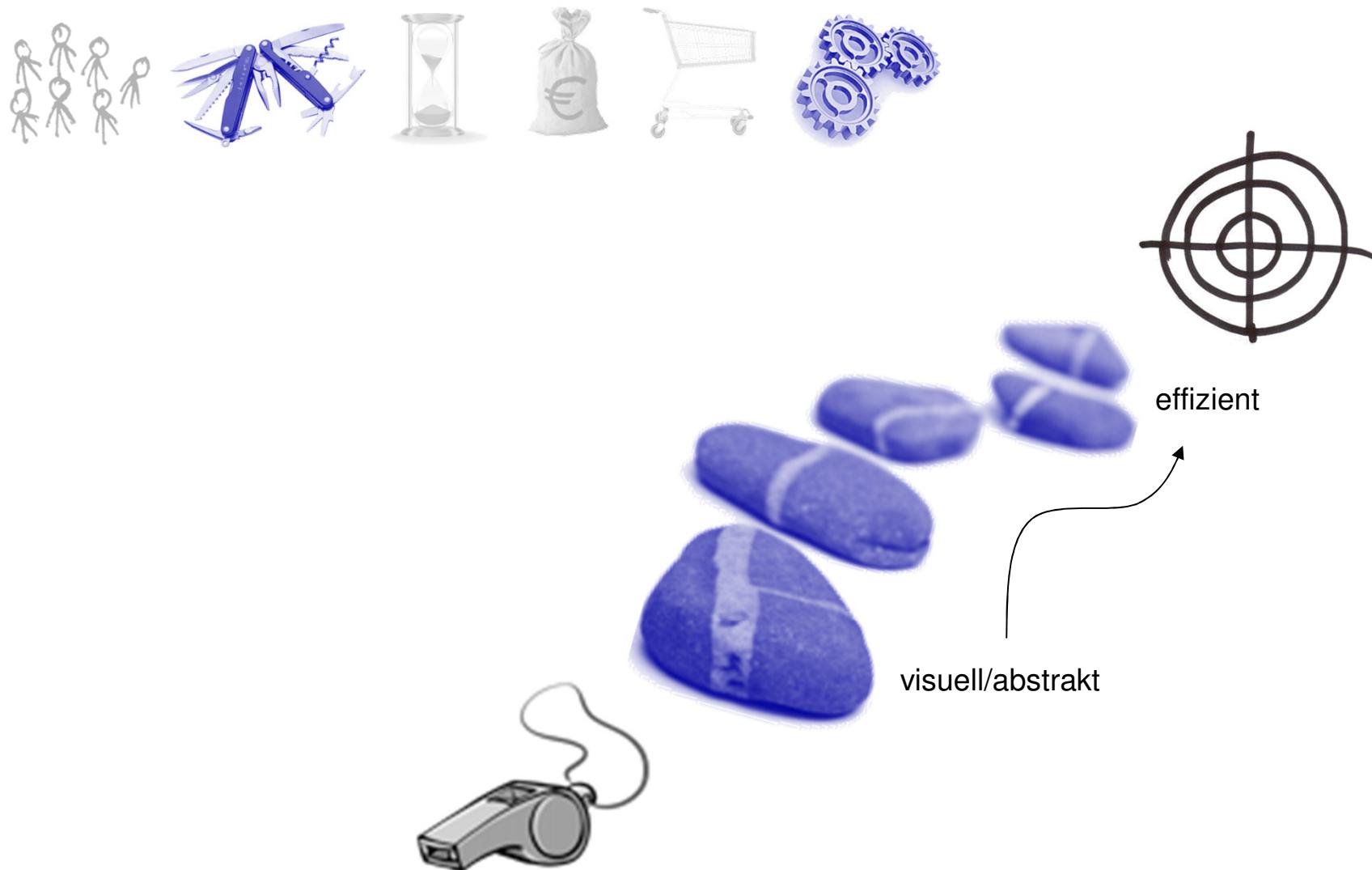
C/C++



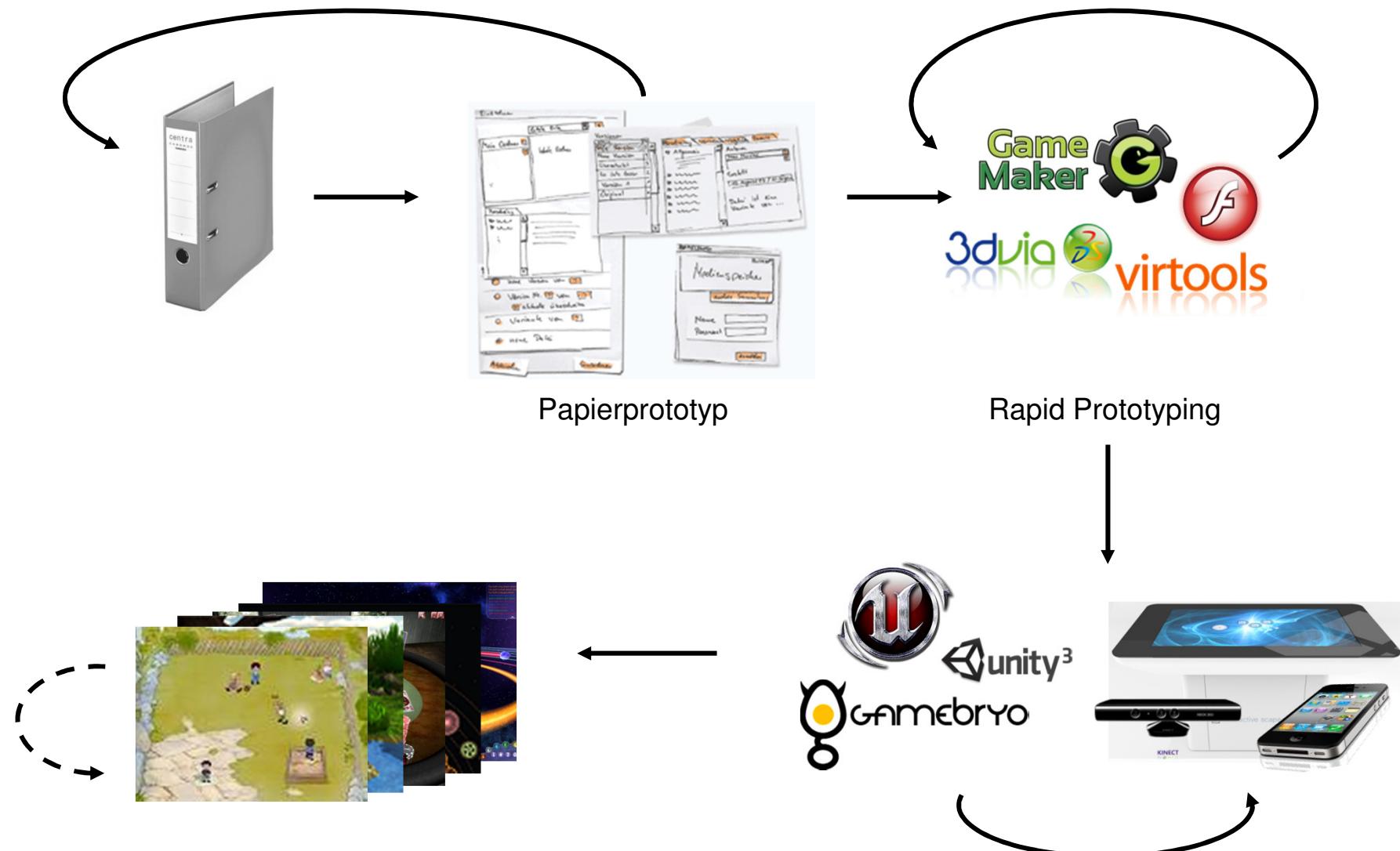
Softwareentwicklungsprozess



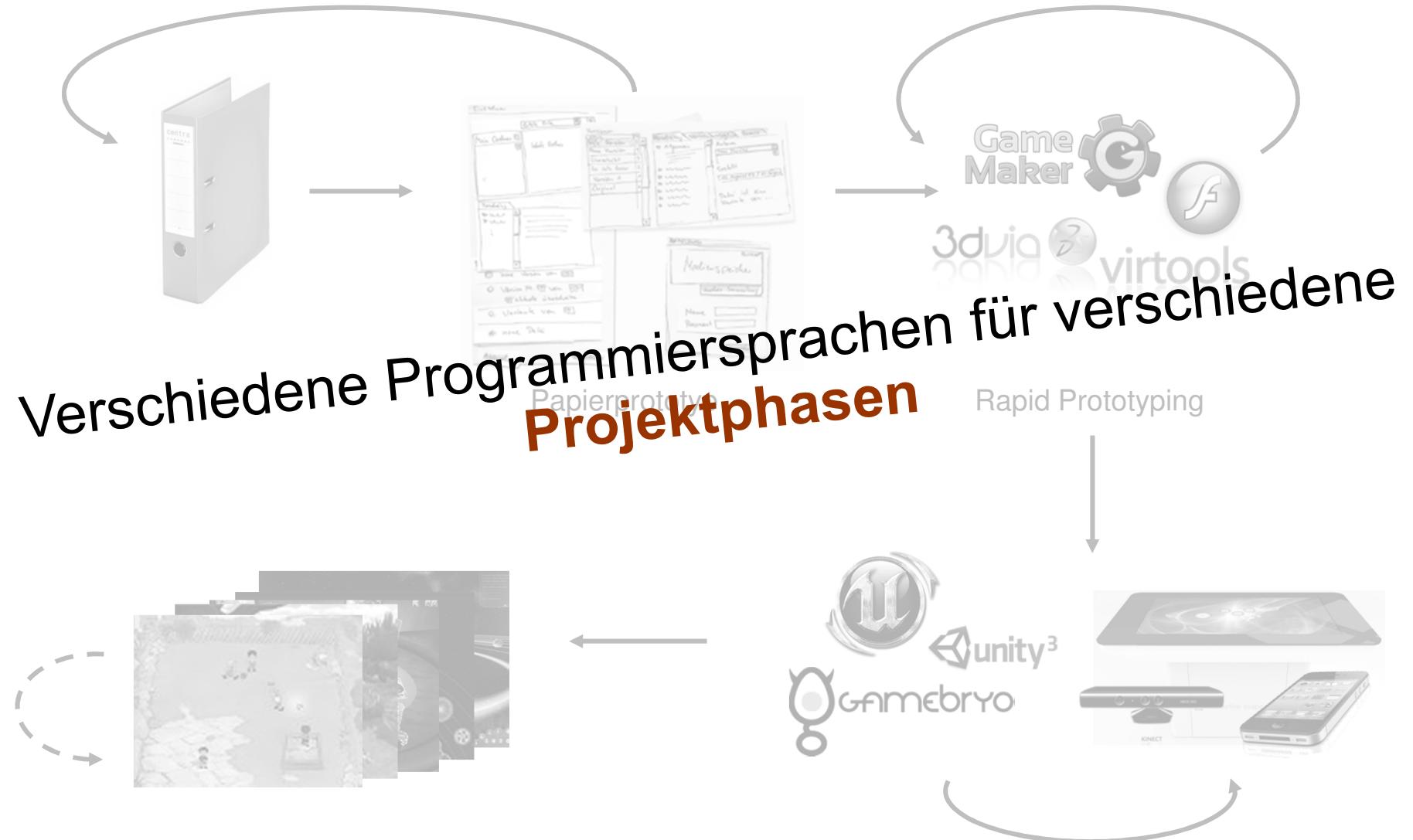
Softwareentwicklungsprozess



Phasen der Softwareentwicklung: Spieleindustrie



Phasen der Softwareentwicklung: Spieleindustrie



Wie wähle ich die passende Programmiersprache
für ein Projekt?

Zusammenfassung

- Von Programmiersprachen auf Paradigmen abstrahieren
- Alle Paradigmen sind gleich mächtig
- Programmierparadigmen werden kombiniert
- Programmiersprachen werden kombiniert
- Betriebssysteme werden kombiniert
- Verschiedene Programmiersprachen kommen in einem Softwareprojekt in verschiedenen Phasen zum Einsatz

- Flexibilität und breites Verständnis werden also erwartet!

Was sind die Rahmenbedingungen
für unser gemeinsames Projekt?

Unsere wichtigsten Themen und Ziele

Grundlagen der Berechenbarkeit/Programmiermethodik

- Syntax und operationelle Semantik imperativer Programmiersprachen
- Einführung und Ausbildung zum Selbststudium
- Lernen objektorientiert zu denken und zu programmieren

Einführung in die Laufzeit- und Komplexitätsanalyse

- Landau-Symbolik
- Kleine Programmabschnitte analysieren und vergleichen können
- Effiziente Lösungen für neuartige Probleme formulieren

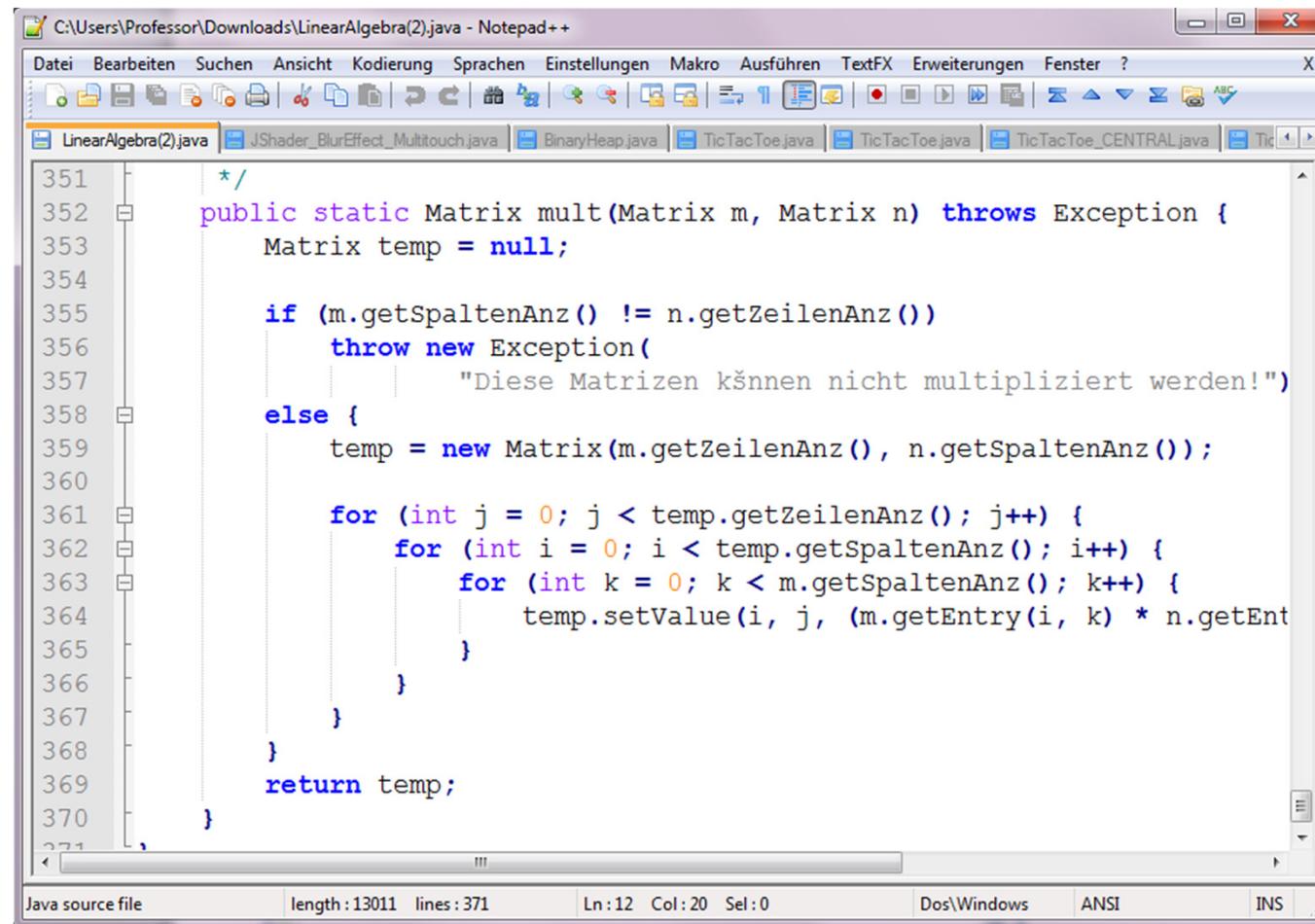
Formale Verfahren zur Spezifikation und Verifikation imperativer Programme bzw. Testtechniken

- Programme möglichst fehlerfrei schreiben
- Entworfene Programme testen
- Test-Driven-Development beherrschen

Java

Entwicklungsumgebung für Java (erste Tage)

Für die ersten kleinen Programme in Java werden wir Notepad++ oder einen vergleichbaren Editor mit Syntaxhighlighting verwenden:



The screenshot shows a window of Notepad++ displaying a Java source file named `LinearAlgebra(2).java`. The code implements a static method `mult` for multiplying two matrices. The code uses nested loops to iterate through rows and columns, and checks if the number of columns in the first matrix equals the number of rows in the second matrix. If not, it throws an exception. Otherwise, it creates a new matrix and fills it with the result of the multiplication. The code is annotated with line numbers from 351 to 370.

```
351 */  
352     public static Matrix mult(Matrix m, Matrix n) throws Exception {  
353         Matrix temp = null;  
354  
355         if (m.getSpaltenAnz() != n.getZeilenAnz())  
356             throw new Exception(  
357                 "Diese Matrizen können nicht multipliziert werden!");  
358         else {  
359             temp = new Matrix(m.getZeilenAnz(), n.getSpaltenAnz());  
360  
361             for (int j = 0; j < temp.getZeilenAnz(); j++) {  
362                 for (int i = 0; i < temp.getSpaltenAnz(); i++) {  
363                     for (int k = 0; k < m.getSpaltenAnz(); k++) {  
364                         temp.setValue(i, j, (m.getEntry(i, k) * n.getEnt  
365                         )  
366                     }  
367                 }  
368             }  
369             return temp;  
370         }  
371     }
```

Java source file length : 13011 lines : 371 Ln : 12 Col : 20 Sel : 0 Dos\Windows ANSI INS

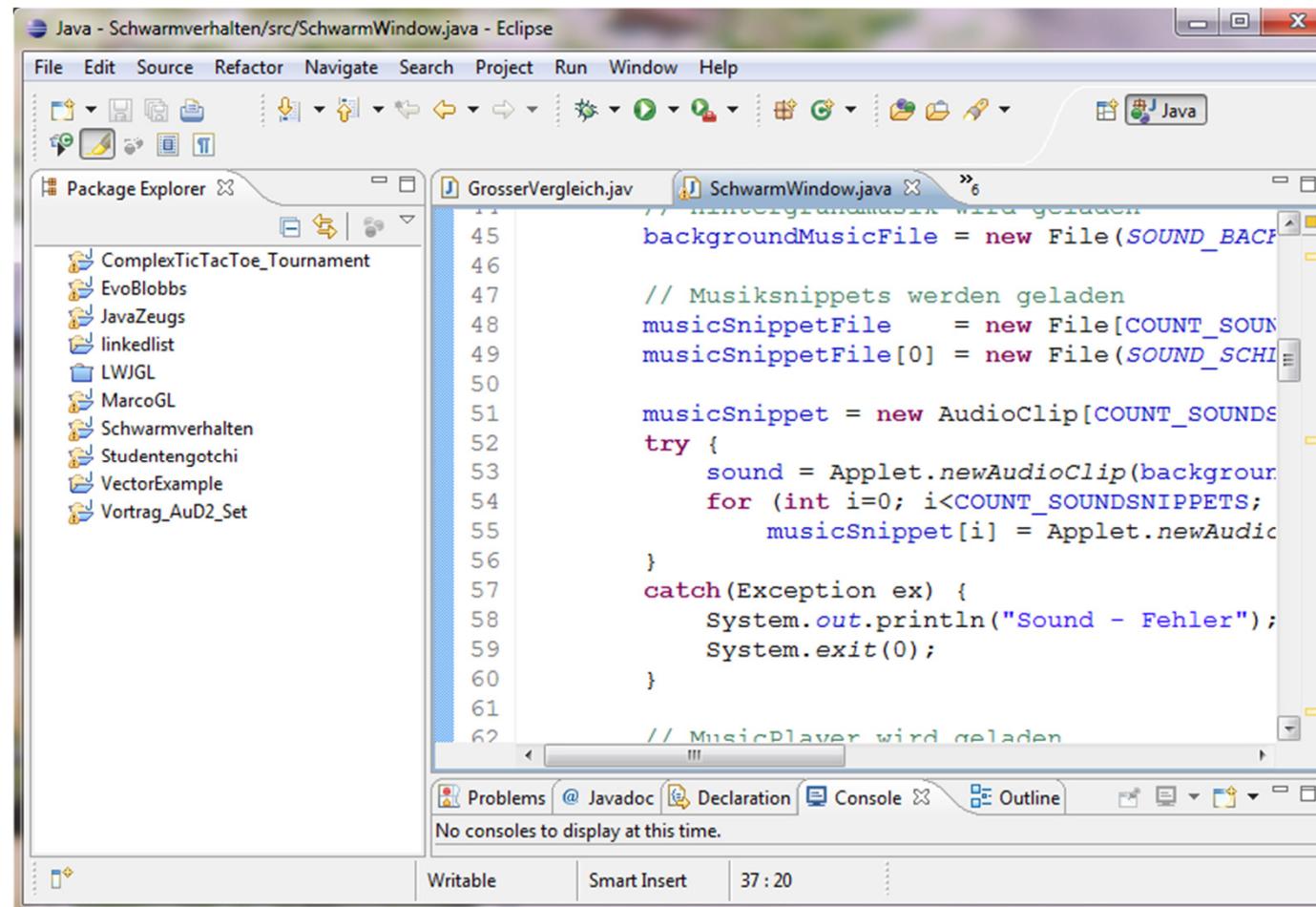
Live-Coding-Session

```
number": $number,
contents": $contents,
$count": $count,

$i + 1' . $totalSecurity)
echo "checked";
if ($i == 0) {
    echo "checked");
}
```

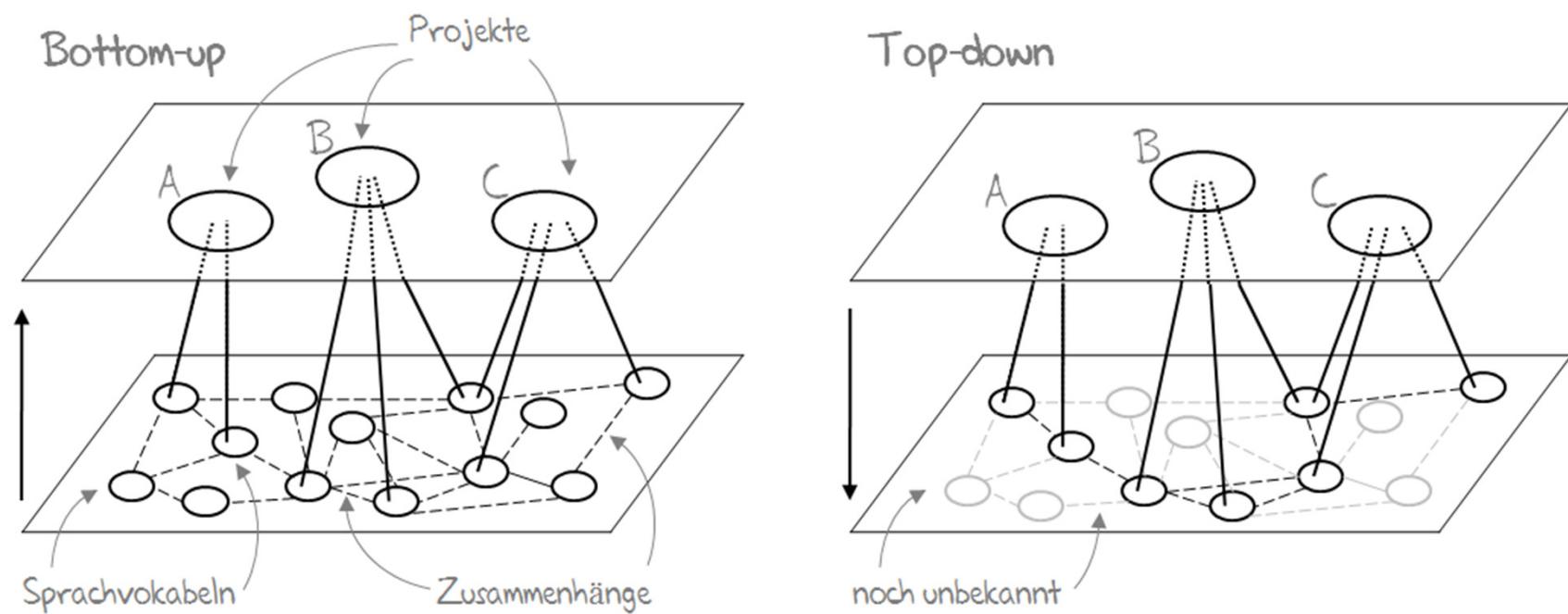
Entwicklungsumgebung für Java

Für eine professionelle Entwicklung größerer Programme in Java werden wir später lernen mit Eclipse zu arbeiten:



Philosophie der Veranstaltung

Top-Down- versus Bottom-Up-Lernen

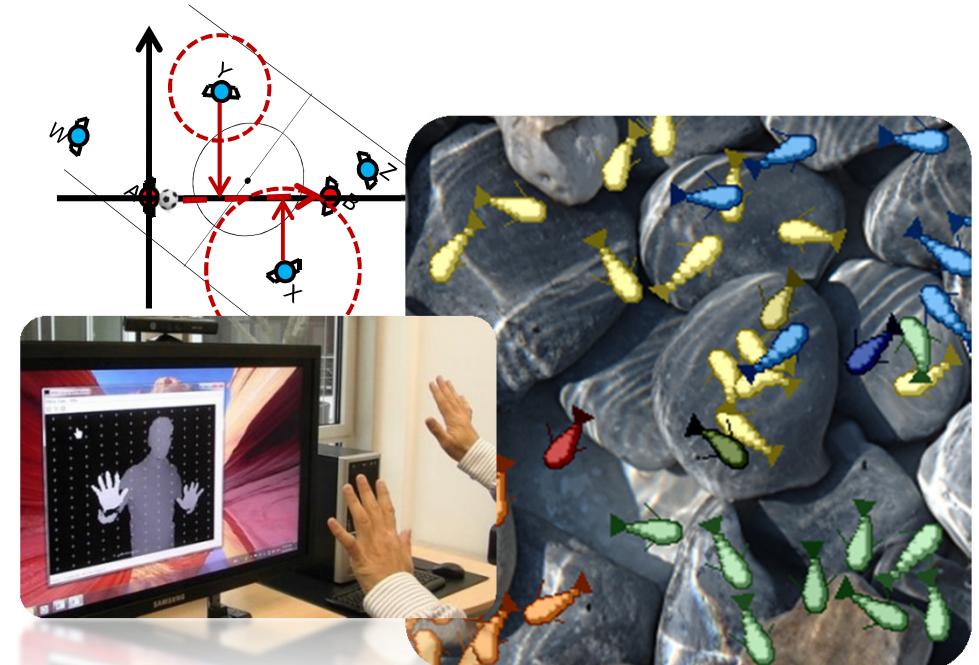


Motivation zu Projekten und Ausblick

Programmieren lernen heißt eine Sprache lernen ... und Sprachen muss man sprechen!

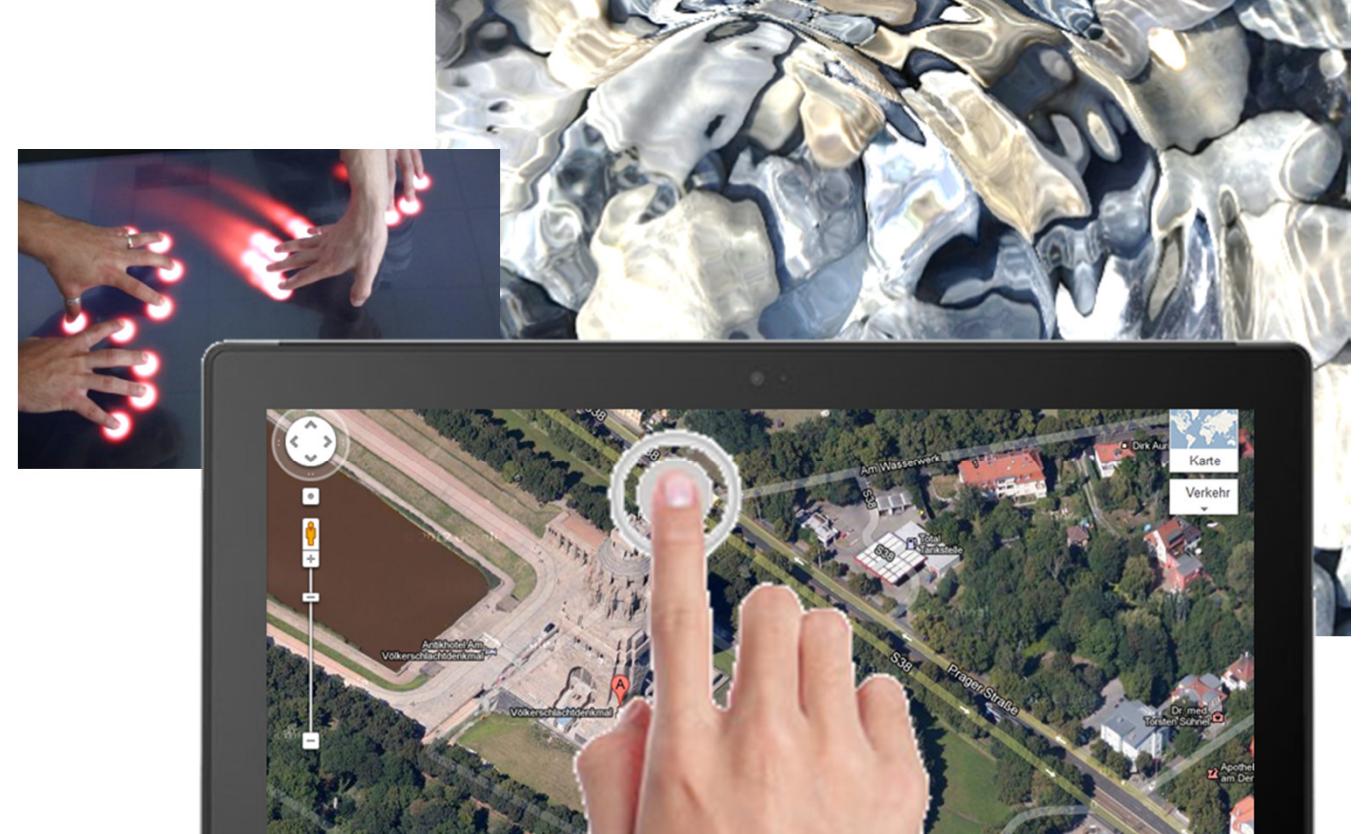
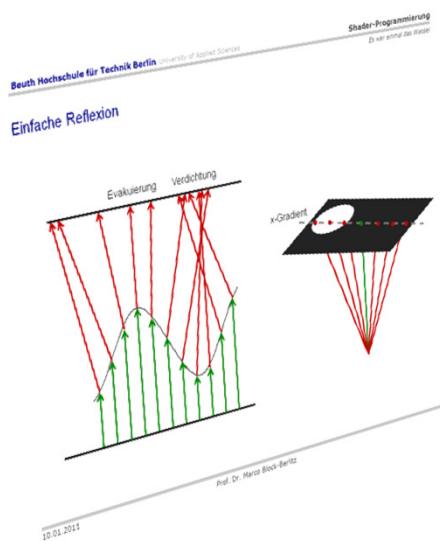
Das bedeutet für uns:

- Projekte
- Projekte
- Projekte
- ...
- viele Projekte



Smartphone- und Tabletprojekte mit Java

Klein, multitouchfähig ... jederzeit dabei!



Einstieg in die Softwaretechnik

Visuelle Sprachen werden oft für die Kommunikation eingesetzt ...

Wir werden diese in den verschiedenen Phasen der Softwareentwicklung kennenlernen und einsetzen.



Noch eine kleine Anmerkung ...



Dialog ausdrücklich erwünscht

Vorlesungsteil

Vorbereitungen und Javas kleinste Bausteine



Die Basis einer gesunden Ordnung ist ein großer Papierkorb.
Kurt Tucholsky

Übersicht zum Vorlesungsinhalt

zeitliche Abfolge und Inhalte können variieren

Vorbereitungen und Javas kleinste Bausteine

- Motivation zu Java
- Installation der aktuellen Java-Version
- Primitive Datentypen und ihre Wertebereiche
- Wiederholung: Boolesche Algebra
- Variablen und Konstanten
- Operationen auf primitiven Datentypen
- Umwandlungen von Datentypen



Warum gerade mit **Java** beginnen?

Warum gerade mit Java beginnen?

Es gibt viele Gründe mit dem Programmieren zu beginnen, aber warum sollte es gerade die Programmiersprache Java sein?

- Java ist leicht zu erlernen
- relativ kleiner Befehlsumfang
- strikte Typsicherheit
- plattformunabhängig
- hat dominierende Stellung auf mobilen Geräten und in der Welt der Server
- mit dem Java Native Interface (JNI) können Windows-DLLs und C++-Bibliotheken eingebunden werden
- sehr große Community

Generell gilt: Die Chance in Java etwas falsch zu machen ist sehr viel kleiner als in anderen Programmiersprachen, wie z.B. in C oder C++.

Java wird deshalb auf Grund dieser Eigenschaften gerade in der Lehre intensiv eingesetzt.

Was leistet Java nicht?

Es gibt aber auch Systeme, bei denen Java nicht eingesetzt werden sollte bzw. folgende Nachteile:

- Geschwindigkeitsrelevante bzw. hardwarenahe Softwaresysteme
- Operatoren können nicht überladen werden

In C++ können wir beispielsweise den Operator + für Vektoren überladen:

```
Vektor c = a + b;
```

In Java können wir dafür nur eine Funktionsschreibweise anbieten:

```
Vektor c = Vektor.add(a, b);
```

Installation von Java und Vorbereitungen

Wir installieren die aktuelle Java-Version 1.7 mit interessanten Neuerungen.

- 1) Download¹⁾ und Installation des JDK (Java Development Kit)
- 2) Download^{2, 3)} und Installation einer Entwicklungsumgebung
- 3) Testen der installierten Javaversion.



- i) In einer Konsole prüfen, ob der Compiler vorhanden ist:

```
c:\javac
```

- ii) Folgendes Testprogramm im Editor schreiben und mit dem Namen **TestProgramm.java** speichern:

```
public class TestProgramm {}
```

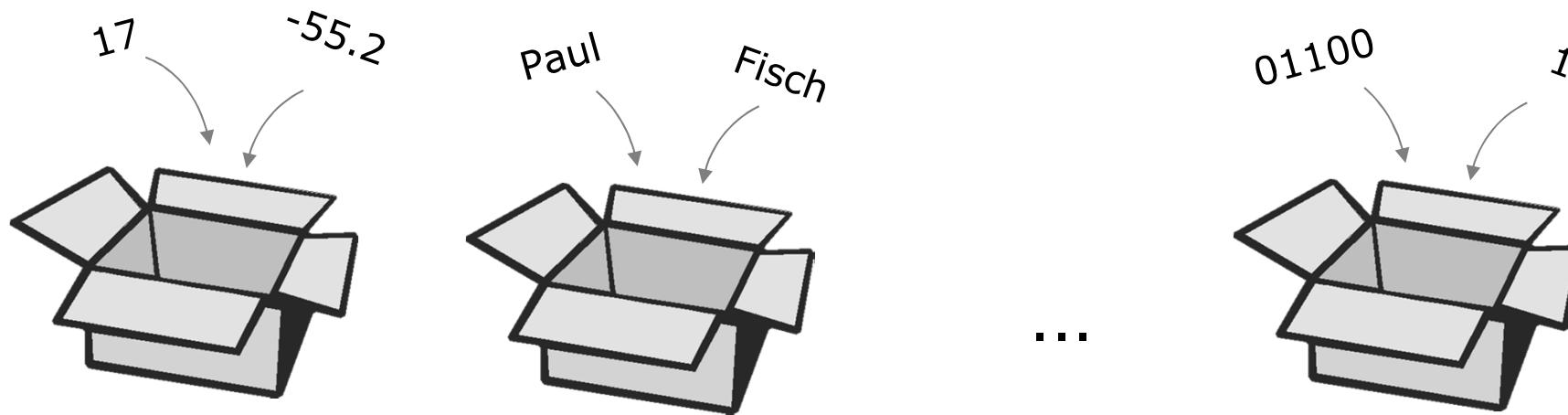
- iii) Kompilieren und ausführen des Testprogramms:

```
c:\javac TestProgramm.java
```

```
c:\java TestProgramm
```

1) Oracle: <http://www.oracle.com/de/technologies/java/index.html>
2) Notepad++: <http://notepad-plus-plus.org>
3) Eclipse: <http://www.eclipse.org>

Variablen als Speicherplatz



Primitive Datentypen in Java

Wahrheitswerte	Zeichen und Symbole	Zahlen
boolean	char	byte, short, int, long, float, double

Warum haben **Zahlen** eine besondere Stellung?

Primitive Datentypen und ihre Wertebereiche

Datentyp	Wertebereich	Bit
boolean	true, false	8
char	0 ... 65.535 (z.B. 'a', 'b', ..., 'A', ..., '1', '2', ...)	16
byte	-128 bis 127	8
short	-32.768 bis 32.767	16
int	-2.147.483.648 bis 2.147.483.647	32
long	-9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807	64
float	+/-1,4E-45 bis +/-3,4E+38	32
double	+/-4,9E-324 bis +/-1,7E+308	64

Deklaration von Variablen

Deklaration von Variablen:

<Datentyp> <Name>;

Beispiel:

```
boolean a;
```



Zuweisung von Werten

Nach der Deklaration steht die Variable bereit und kann einen Wert mit dem Symbol = zugewiesen bekommen.

Die Zuweisung erlaubt aber nur den Wert der rechten Seite auf die linke zu übertragen:

```
a = true;
```

Es andersherum aufzuschreiben wäre fehlerhaft:

```
true = a;
```

Deklaration und Zuweisung

Deklaration von Variablen und Zuweisung in einem Schritt:

<Datentyp> <Name> = <Wert>;

Beispiele:

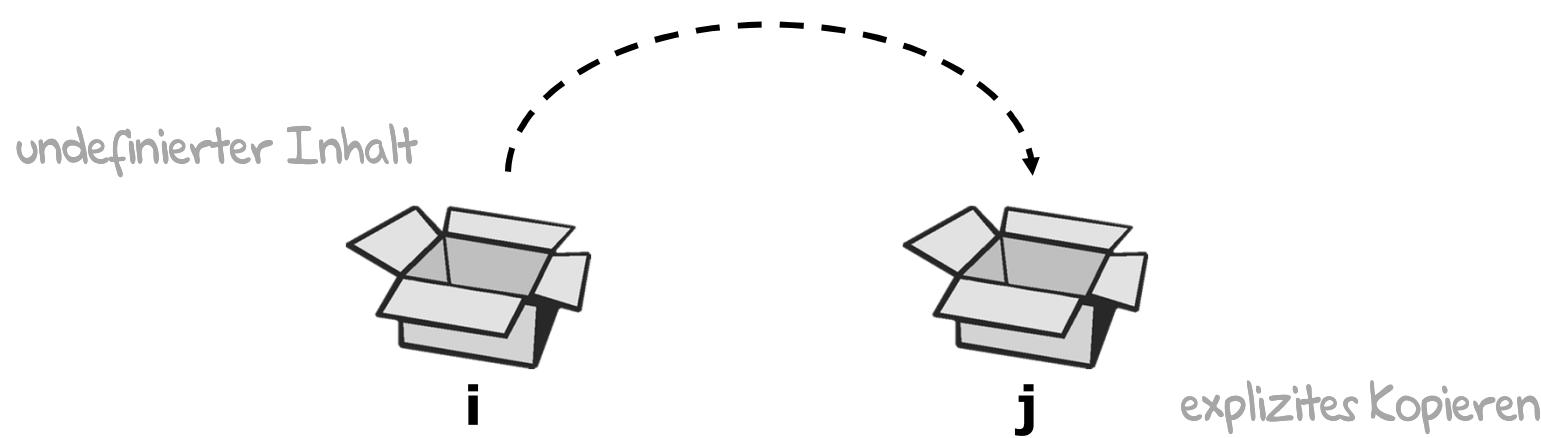
```
boolean a, b;  
a = true;  
  
char c = 'b';  
  
int d = 12, e, f = 0;
```

Deklaration und Initialisierung vor Verwendung!

Variablen müssen vor dem Einsatz immer initialisiert (also mit einem Wert versehen) werden.

Folgendes Beispiel führt zu einem Fehler:

```
int i;  
int j = i;
```



Bezeichnungen von Variablen

Es gibt zwar ein paar Beschränkungen für die Namensgebung, aber der wichtigste Grund für die Wahl ist die Aufgabe also der Zweck der Variable.

Ein paar Beispiele dazu:

```
boolean spielläuft;  
double piViertel;  
int anzIterationen;
```

Variablennamen beginnen mit einem Buchstaben, Unterstrich oder Dollarzeichen, nicht erlaubt sind dabei Zahlen. Nach dem ersten Zeichen dürfen aber sowohl Buchstaben als auch Zahlen folgen. Operatoren, Schlüsselwörter und vordefinierte Literale dürfen ebenfalls nicht als Variablennamen verwendet werden.

Zu den reservierten Schlüsselwörtern und Literalen in Java gehören die folgenden:

abstract, assert, boolean, break, byte, case, catch, char, class,
const, continue, default, do, double, else, enum, extends, false,
final, finally, float, for, future, generic, goto, if, implements,
import, inner, instanceof, int, interface, long, native, new, null,
operator, outer, package, private, protected, public, rest, return,
short, static, strictfp, super, switch, synchronized, this, throw,
throws, transient, true, try, var, void, volatile und while

Syntaxhervorhebung

Schlüsselwörter werden fett und Kommentare kursiv gekennzeichnet, das fördert Übersicht und Lesbarkeit.

Hier ein Beispiel ohne Syntaxhervorhebung (Syntaxhighlighting):

```
// zählt die Anzahl der bereits besetzten Felder
private int countSigns(int[][] b) {
    int count=0;
    for (int i=0; i < 3; i++)
        for (int j=0; j < 3; j++)
            if (b[i][j] !=0)
                count++;
    return count;
}
```

... und hier mit:

```
// zählt die Anzahl der bereits besetzten Felder
private int countSigns(int[][] b) {
    int count=0;
    for (int i=0; i < 3; i++)
        for (int j=0; j < 3; j++)
            if (b[i][j] !=0)
                count++;
    return count;
}
```

Groß- und Kleinschreibung in Java

Bei Java muss auf Groß- und Kleinschreibung geachtet werden, denn unterschiedliche Schreibweisen bezeichnen unterschiedliche Variablen (*case-sensitive*).

Folgendes würde deshalb nicht funktionieren:

```
boolean istFertig;  
istFERTIG = true;
```

Variablenbezeichner und Code-Konventionen

Durch eine **überlegte Variablenbezeichnung** werden zwei Dinge gefördert:

- Zum Einen benötigt der Programmautor weniger Zeit, um seine eigenen Programme zu lesen, deren Erstellung vielleicht schon etwas länger zurück liegt, und
- zum Anderen fördert es die Zusammenarbeit mit anderen Programmierern.

Im Laufe der Zeit haben sich einige **Konventionen**¹⁾ bezüglich der Erstellung von Programmen etabliert. Der Programmierer sollte sich an diese halten, um sich selbst und anderen das Leben nicht unnötig schwer zu machen. Frei nach dem Motto:

Wir müssen nicht immer alles machen, was erlaubt ist.

Bei der Bezeichnung von Variablen in Java hat sich die Konvention durchgesetzt, dass wir Kombinationen von Verben und/oder Substantiven verwenden.

Die Worte werden dabei, bis auf das erste, beginnen mit einem Großbuchstaben geschrieben (*camelCase*).

1) Oracle: <http://www.oracle.com/technetwork/java/codeconv-138413.html>

Variablen versus Konstanten

Manchmal ist es notwendig, Variablen zu verwenden, die die Eigenschaft besitzen, während des gesamten Programmablaufs unverändert zu bleiben.

Beispielsweise eine Näherung der Zahl **pi**:

```
double pi = 3.14159;
```

Damit aber nun gewährleistet ist, dass der Softwareentwickler selbst oder auch ein anderer weiß, dass dieser Platzhalter im Speicher nicht variabel und demzufolge keine Änderung erlaubt ist, schreiben wir einfach ein **final** davor, verwenden (laut Konvention) für den Namen nur Großbuchstaben und machen so aus einer Variable eine Konstante.

Die Syntax dafür sieht wie folgt aus:

```
final <Datentyp> <Name> [= <Wert>];
```

Jetzt können wir unmissverständlich den Platzhalter für **pi** als Konstante deklarieren:

```
final double PI = 3.14159;
```

Java-Compiler überprüft final

Sollten wir dennoch versuchen, Konstanten zu verändern ...

```
final double PI = 3.14159;  
PI = 4;
```

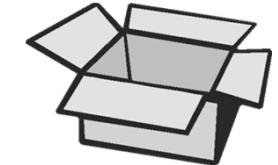
... liefert der Java-Compiler einen Fehler:

```
KonstantenTest:4: cannot assign a value to final variable PI  
        PI = 4;  
        ^  
1 error
```

Primitive Datentypen und ihre Operationen: Datentyp boolean

Der Datentyp boolean bezeichnet einen Wahrheitswert und kann demzufolge **true** oder **false** sein.

```
boolean a;  
a = true;
```



Es gibt eine weit verbreitete Möglichkeit, auf die wir später noch genauer eingehen werden, einen boolean für eine **Entscheidung** zu verwenden:

```
if (<boolean>)  
    <Anweisung>;
```

Wenn der boolean den Wert true hat, führen wir die nachfolgende Anweisung aus, ansonsten überspringen wir diesen Abschnitt.

Das wichtigsten logische Operationen: UND, ODER und NICHT

B_1	B_2	B_1 UND B_2	B_1 ODER B_2	B	NICHT B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	1	0

UND, ODER und NICHT in Java

In Java verwenden wir `&&` (UND), `||` (ODER) und `!` (NICHT):

```
boolean a, b, c;  
a = true;  
b = false;  
c = a && b;
```

Welchen Wert hat c?

```
boolean a, b, c;  
a = true;  
b = a && a;  
c = b || a;
```

Welchen Wert hat c?

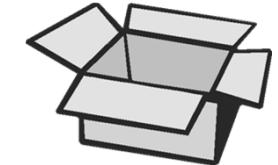
```
boolean a = true, b = false, c, d;  
c = a && b;  
d = (a || b) && !c;
```

Welchen Wert hat d?

Datentyp char

Zeichen oder Symbole werden durch den Datentyp char identifiziert und zwischen ' ' notiert:

```
char d = '7';
char e = 'b';
```



Die Variable d trägt als Inhalt das Zeichen '7' und wird aber nicht als Zahl 7 interpretiert.

Schauen wir uns die relationalen Operatoren (Vergleichsoperatoren) stellvertretend für char an:

```
boolean d, e, f, g;
char a, b, c;
a = '1';
b = '1';
c = '5';

d = a == b;          // gleich
e = a != b;          // ungleich
f = a < c;           // kleiner
g = c >= b;          // größer-gleich
```

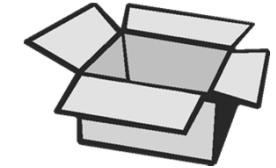
Welche Werte gelten
für d, e, f und g?

Datentyp int

Der Datentyp int hat einen Wertebereich von: $\{-2^{31}, -2^{31}+1, \dots, -1, 0, 1, 2, \dots, 2^{31}-1\}$

Der kleinste bzw. größte darstellbare Wert liegt jeweils in einer Konstanten vor:

```
int minimalerWert = Integer.MIN_VALUE;  
int maximalerWert = Integer.MAX_VALUE;
```



Was würde jetzt passieren, wenn wir zu dem größten Wert eine 2 addieren?

$$2^{31}-1 + 2 = 2147483647 + 2 = \text{Integer.MAX_VALUE} + 2$$

Statt 2147483649 erhalten wir **-2147483647** auf Grund des zyklischen Verhaltens durch den begrenzten Speicher.

Hier der Grund dafür:

$$\begin{array}{r} 1 \dots 111 \\ + 0 \dots 001 \\ \hline 1 0 \dots 000 \end{array} \quad \begin{array}{l} \text{int a} \\ \text{int b} \\ \text{int c} \end{array}$$

Übertragsbit
wird "vergessen"

Stichwort: Zweierkomplement-Darstellung

Zweierkomplement-Darstellung I

Für 0 und positive Zahlen (z.B. Zweierkomplement-Darstellung mit 4 Bit) werden die entsprechenden Binärzahlen wie folgt erstellt:

$$0 = 0000$$

$$+1 = 0001$$

$$+2 = 0010$$

$$+3 = 0011$$

$$+4 = 0100$$

$$+5 = 0101$$

$$+6 = 0110$$

$$+7 = 0111$$

Für negative Zahlen negieren wir die Binärzahl elementeweise und addieren anschließend eine 1 dazu (z.B. $-1 \Rightarrow 0001 \Rightarrow 1110 \Rightarrow 1111$):

$$-1 = 1111$$

$$-2 = 1110$$

$$-3 = 1101$$

$$-4 = 1100$$

$$-5 = 1011$$

$$-6 = 1010$$

$$-7 = 1001$$

$$-8 = 1000$$

Test mit bitweiser Negation

In Java wird die bitweise Negation mit dem Operator `~` durchgeführt. So ergibt sich beispielsweise für die Dezimalzahl `-1` entsprechend:

0001 (Binärzahl von 1)
=> 1110 (elementeweise invertiert)
=> 1111 (um 1 erhöht).

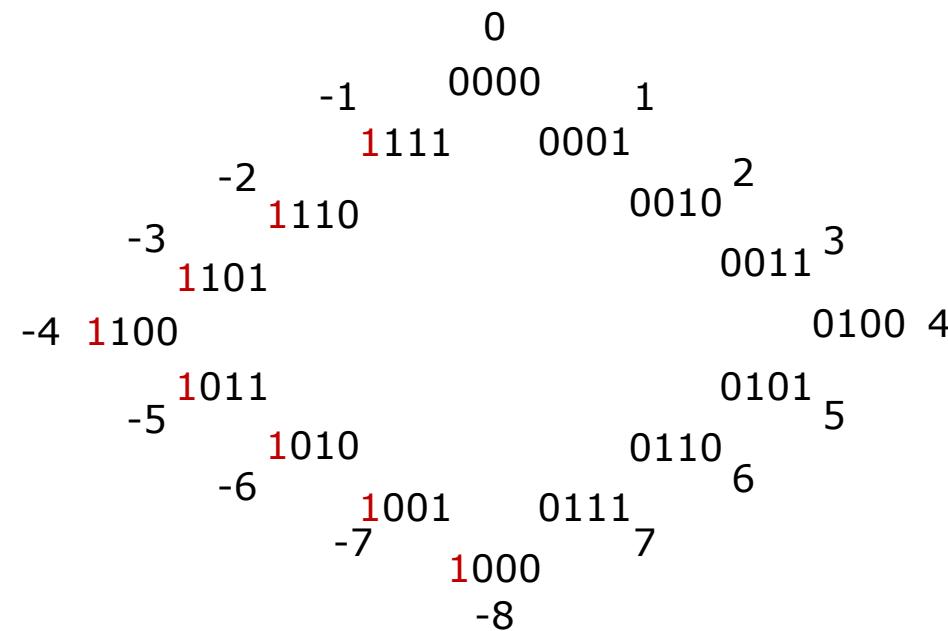
Um das zu prüfen, speichern wir uns in `a` den Wert `1`, invertieren die Binärrepräsentation dieser Zahl elementeweise, erhöhen das Ergebnis anschließend um die Binärdarstellung der `1` und speichern das Ergebnis in `b`:

```
int a = 1;  
int b = ~a + 1;
```

Bei der Ausgabe von `b` erhalten wir in Dezimaldarstellung den Wert `-1`, es hat funktioniert.

Zweierkomplement-Darstellung I0

Folgendes zyklisches Verhalten ergibt sich:



Jetzt ist auch ersichtlich, warum es eine negative Zahl mehr als positive gibt.