BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
LEIPZIG
UNIVERSITY OF COOPERATIVE EDUCATION

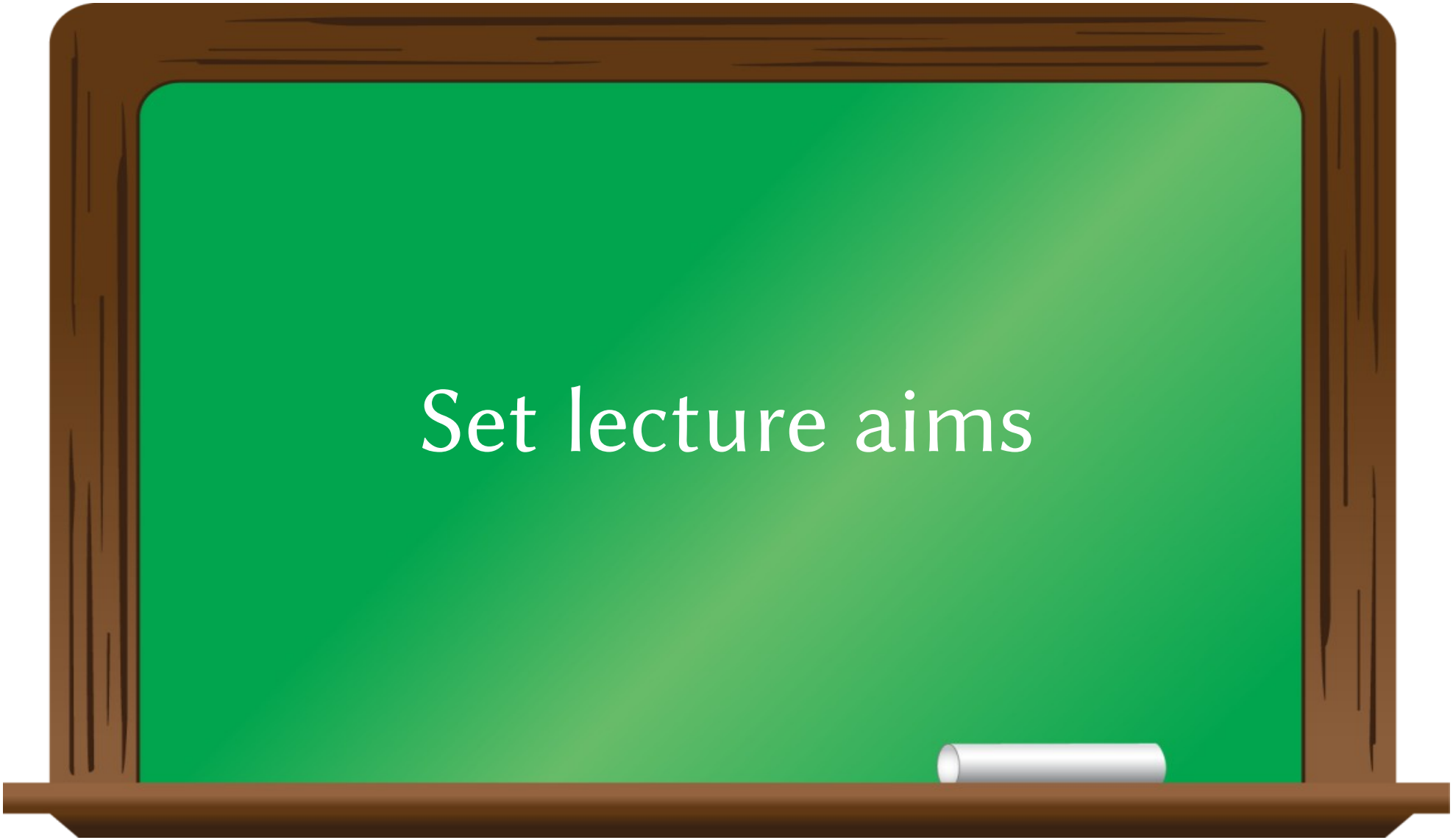
Computer Programming (CP)

Prof. Dr.-Ing. Christian Heller
<christian.heller@ba-sachsen.de>

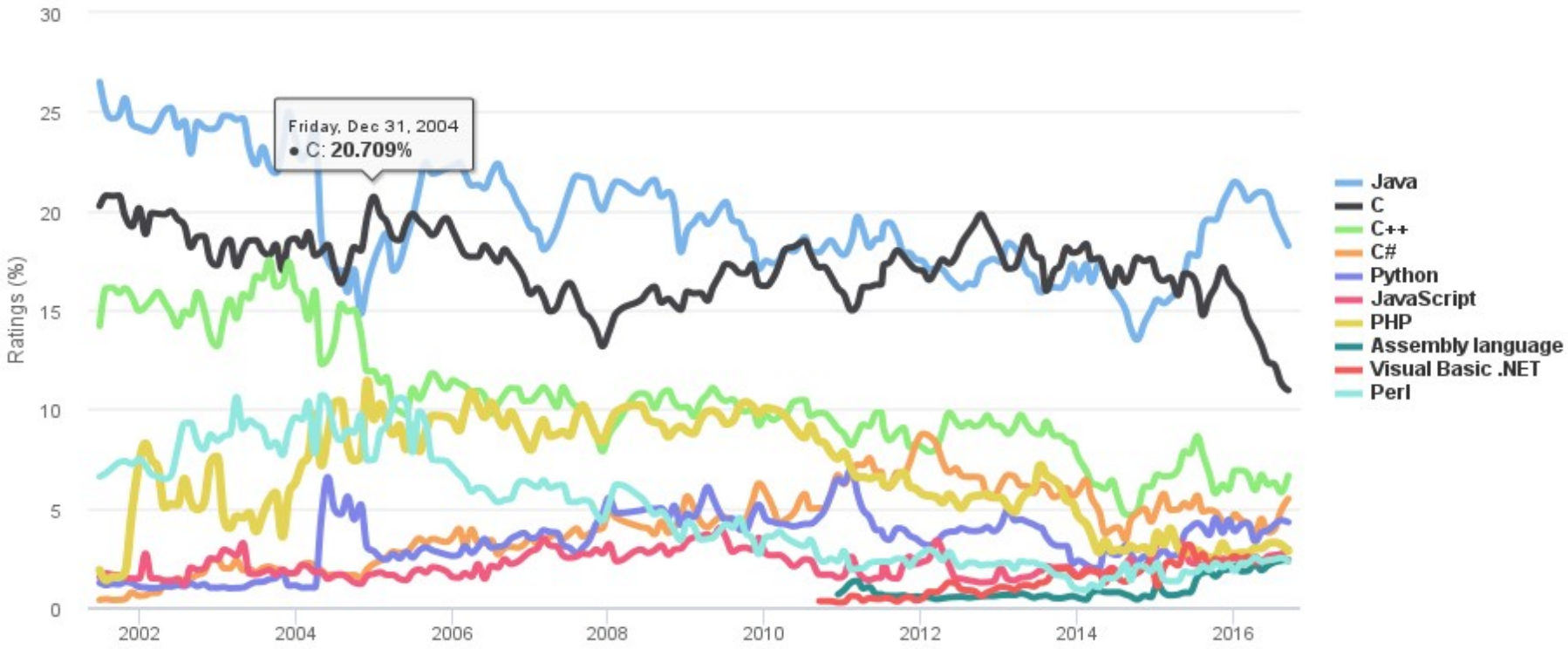
Contents

Contents

Motivation



Programming Community Index



http://www.tiobe.com/

Ranking

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	^	JavaScript	2.929%	+0.59%
7	6	v	PHP	2.847%	+0.32%
8	11	^	Assembly language	2.417%	+0.61%
9	8	v	Visual Basic .NET	2.343%	+0.28%
10	9	v	Perl	2.333%	+0.43%
11	13	^	Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16	^	Swift	1.930%	+0.74%
14	10	vv	Objective-C	1.849%	+0.03%
15	17	^	MATLAB	1.826%	+0.65%
16	34	^^	Groovy	1.818%	+1.31%
17	14	v	Visual Basic	1.761%	+0.23%
18	19	^	R	1.684%	+0.64%
19	44	^^	Go	1.625%	+1.37%

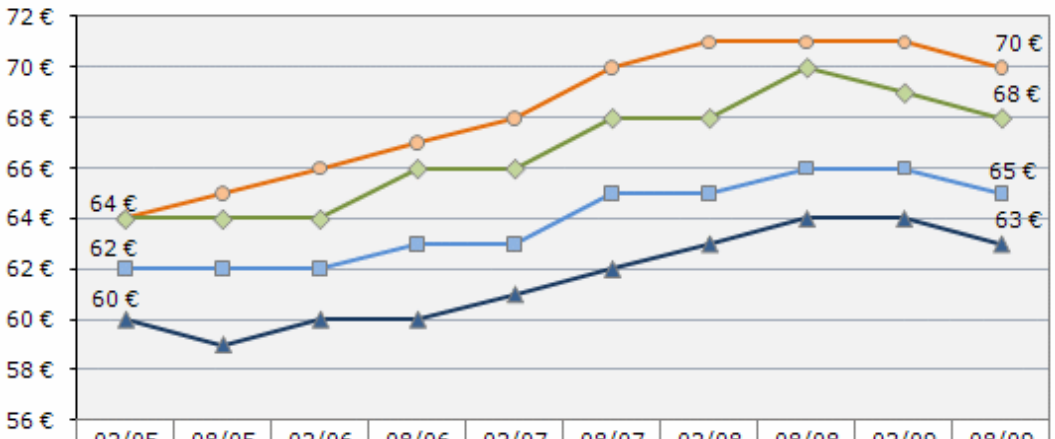
http://www.tiobe.com/

Long-Term History

Programming Language	2016	2011	2006	2001	1996	1991	1986
Java	1	1	1	3	14	-	-
C	2	2	2	1	1	1	1
C++	3	3	3	2	2	2	5
C#	4	5	6	11	-	-	-
Python	5	6	7	24	23	-	-
PHP	6	4	4	8	-	-	-
JavaScript	7	9	8	7	19	-	-
Visual Basic .NET	8	30	-	-	-	-	-
Perl	9	8	5	4	3	-	-
Ruby	10	10	18	32	-	-	-
Lisp	27	12	12	15	7	5	3
Ada	28	16	15	16	6	3	2

http://www.tiobe.com/

Hourly Rate

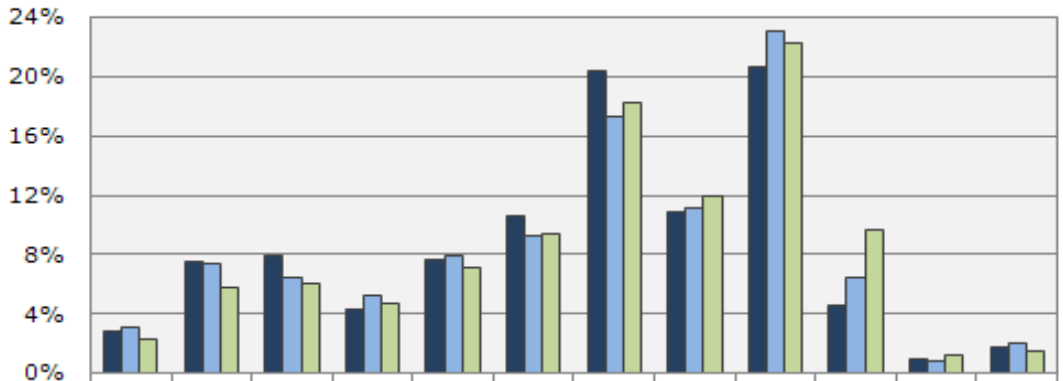


	02/05	08/05	02/06	08/06	02/07	08/07	02/08	08/08	02/09	08/09
Alle IT-FB	64 €	65 €	66 €	67 €	68 €	70 €	71 €	71 €	71 €	70 €
Java	64 €	64 €	64 €	66 €	66 €	68 €	68 €	70 €	69 €	68 €
C++	62 €	62 €	62 €	63 €	63 €	65 €	65 €	66 €	66 €	65 €
C#	60 €	59 €	60 €	60 €	61 €	62 €	63 €	64 €	64 €	63 €

Stundensatzforderungen (in Euro) der IT-Freiberufler in den Bereichen Java, C++ und C# sowie aller in die GULP Profiledatenbank eingetragenen externen IT-Spezialisten; Quelle: GULP Stundensatz Kalkulator und GULP Stundensatz-Auswertung

Regional Distribution

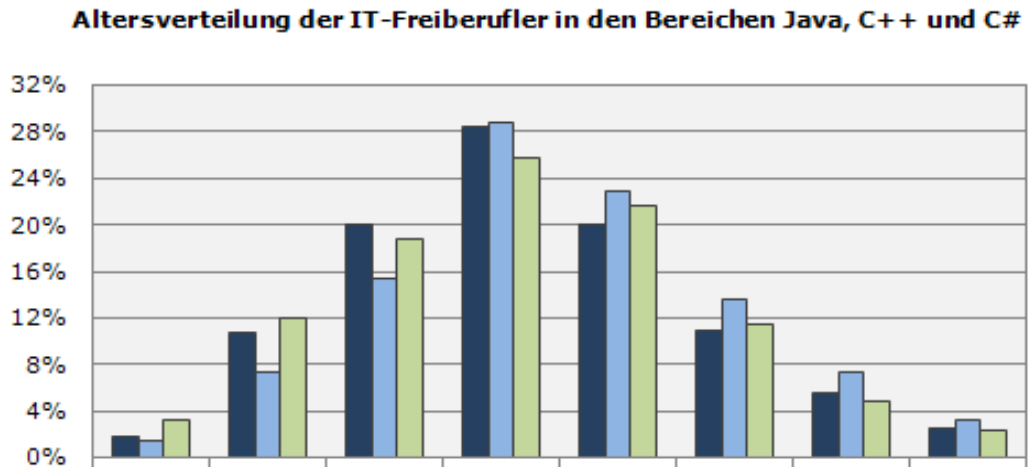
Regionale Verteilung der Projektanfragen zu Java, C++ und C#



	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	A	CH
Java	2,8%	7,5%	7,9%	4,3%	7,7%	10,6%	20,4%	10,8%	20,7%	4,6%	0,9%	1,8%
C++	3,1%	7,4%	6,5%	5,2%	7,9%	9,3%	17,3%	11,1%	23,1%	6,4%	0,8%	2,0%
C#	2,3%	5,8%	6,1%	4,7%	7,1%	9,4%	18,2%	12,0%	22,2%	9,6%	1,2%	1,5%

Anteil (in Prozent) derjenigen Projektanfragen zu Java, C++ oder C#, die in den letzten zwölf Monaten an IT-Freiberufler in der jeweiligen Region gingen.
Quelle: GULP Trend Analyzer

Age Distribution



	<30	30-34	35-39	40-44	45-49	50-54	55-59	>60
Java	1,8%	10,8%	20,0%	28,4%	20,1%	10,9%	5,5%	2,5%
C++	1,5%	7,3%	15,3%	28,7%	22,9%	13,5%	7,4%	3,2%
C#	3,2%	12,0%	18,8%	25,7%	21,6%	11,4%	4,8%	2,3%

Prozentuale Altersverteilung der IT-Freiberufler in den Bereichen Java, C++ oder C#.
Quelle: GULP Trend Analyzer

Programming in Ten Years

"Researchers ... have shown it takes about ten years to develop expertise in any of a wide variety of areas, including chess playing, music composition, telegraph operation, painting, piano playing, swimming, tennis, and research in neuropsychology and topology. The key is deliberative practice: not just doing it again and again, but challenging yourself with a task that is just beyond your current ability, trying it, analyzing your performance while and after doing it, and correcting any mistakes. Then repeat. And repeat again. ..."

Peter Norvig: Teach yourself programming in ten years. 2001
http://www.norvig.com/21-days.html

Recipe for Programming Success [Peter Norvig]

- Get interested in programming, and do some because it is fun
- Talk to other programmers; read other programs
- Program -- the best kind of learning is learning by doing
- Put in some years at a graduate school to get deeper understanding
- Work on projects with other programmers, in various roles
- Be involved in understanding a program written by someone else
- Learn different programming concepts and languages
- Know how long it takes your computer to execute an instruction
- Get involved in language standardization, e.g. style guide writing

Learning Objectives

... to enable students to:

- apply basic elements of higher programming languages
- develop software programs independently
- use Java syntax
- create data structures and algorithms
- understand object-oriented architectures
- manage exceptions

Target Group

... students of computer science

Prerequisites

... basic knowledge in hard- and software

Software

- GNU/Linux with KDE Desktop
- current version of Java Development Kit (JDK)
- Integrated Development Environment (IDE) like Eclipse

Curriculum

Term	Hours	Subject	Role
1	28	Web Technologies	Web Developer
1	50	Computer Programming	Software Programmer
2	54	Data Processing	Application Developer
3	48	User Interaction	User Interface Designer
4	42	Software Engineering	Software Architect
4	35	Project Management	Project Manager
5	5 + 60	Software Project	Team Worker
5	60	Programming C/C++	Systems Engineer
5	60	CYBOP	Knowledge Modeller
6	56	Server Side Technologies	Enterprise Architect

Schedule

- Revision: on blackboard (30 min)
- Lecture: new matter (60 min)
- Break (30 min)
- Presentation: done by students (2 x 40 min)
- Break (30 min)
- Exercise: solving example tasks (90 min)
- Self-Study (unlimited, > 50 % in a study ;-)

Literature – Oracle

- Ina Brenner. Das große SCJP Trainingsbuch. Franzis Professional Series. ISBN-13: 978-3-7723-702908
- James Gosling, Bill Joy, Guy Steele, Gilad Bracha: The Java Language Specification. Third Edition. Boston: Addison-Wesley, 2005. http://java.oracle.com/docs/books/jls/download/langspec-3.0.pdf
- Java Documentation. http://java.oracle.com/javase/6/docs/
- Java API. http://java.oracle.com/javase/6/docs/api/
- Java Tutorials. http://java.oracle.com/docs/books/tutorial/index.html
- Java Training. http://java.oracle.com/developer/onlineTraining/
- Java Code Samples. http://java.oracle.com/developer/codesamples/index.html

Literature - Basics

- Guido Krüger. Handbuch der Java-Programmierung. Studentenausgabe. Addison-Wesley, 2006
- Christian Ullenboom. Java ist auch eine Insel. Programmieren mit der Java Standard Edition Version 5 / 6. Galileo Press, 2006
- Dietmar Abts. Grundkurs Java. vieweg, 2008
- Cay S. Horstmann. Core Java: 1. Prentice Hall International, 2007
- Cay S. Horstmann. Core Java: 2. Advanced Features. Prentice Hall, 2008
- Thomas Künneth. Einstieg in Eclipse 3.3: Einführung, Programmierung, Plug-In-Nutzung. Galileo Press, 2007
- Project Euler. Platform offering a series of challenging mathematical/ computer programming problems. http://projecteuler.net/

Literature - OOP and Advanced

- Alexander Niemann. Das Einsteigerseminar Objektorientierte Programmierung in Java. Der methodische und ausführliche Einstieg. Vmi Buch, 2006
- Bernhard Lahres. Praxisbuch Objektorientierung. Von den Grundlagen zur Umsetzung. Galileo Press, 2006
- Stephan Niedermeier. Java und XML. Grundlagen, Einsatz, Referenz. Bonn: Galileo Press, 2006
- Robert Sedgewick. Algorithmen in Java. Teil 1-4. Pearson Studium, 2003

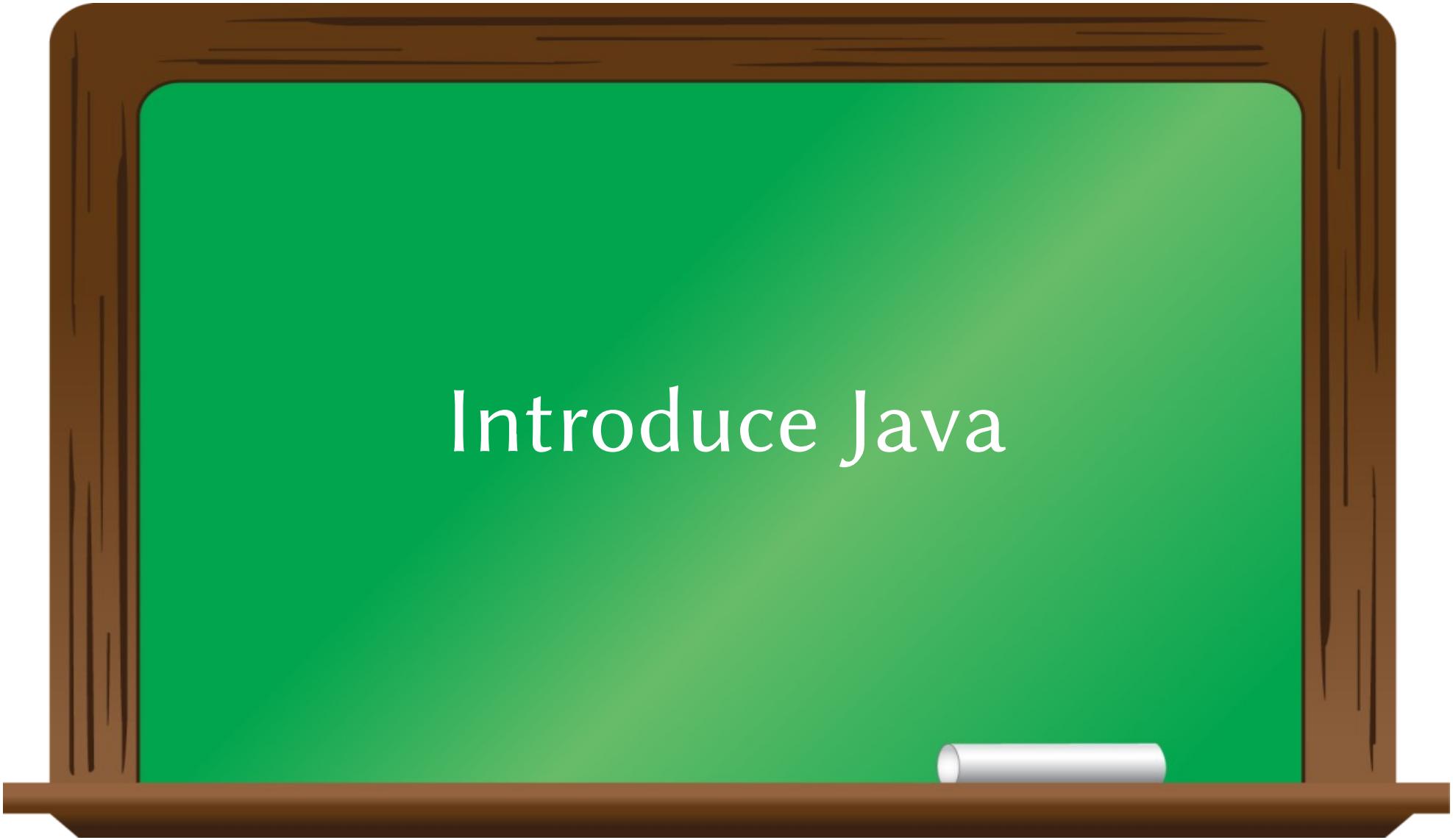
Examination

- presentation during semester as prerequisite
- examination at end of semester 180 min
  - 120 min CP
  - 60 min WT
- practical work on computer
- theory required to understand tasks
- apply fundamental programming techniques

Show example examination!

Contents

Motivation



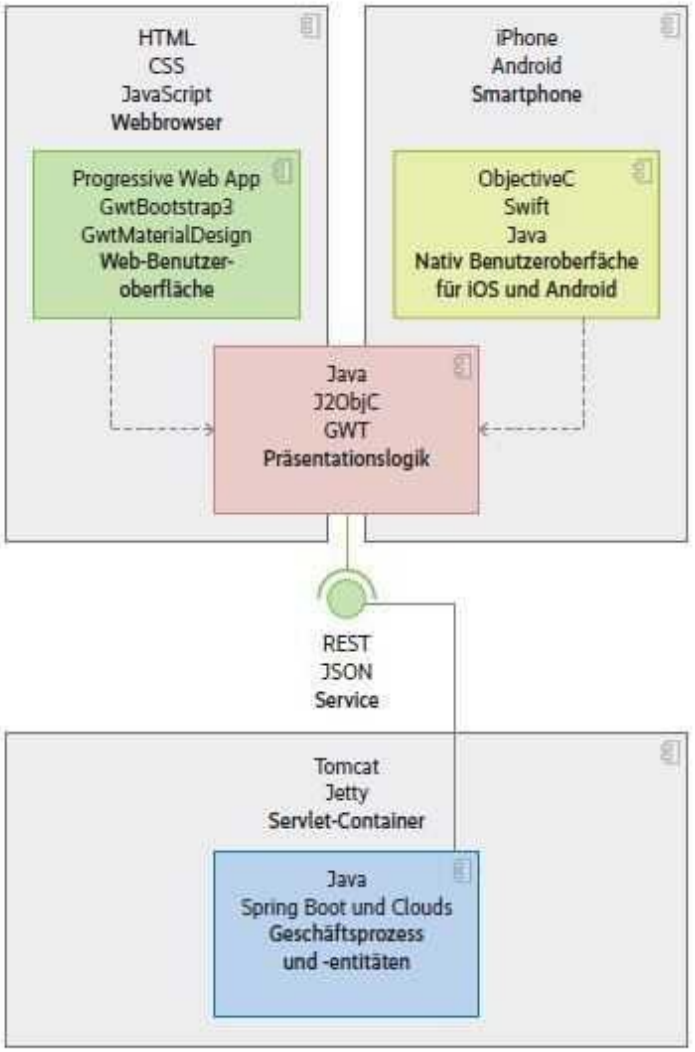
Java Programming Language

↳ Objektorientiert

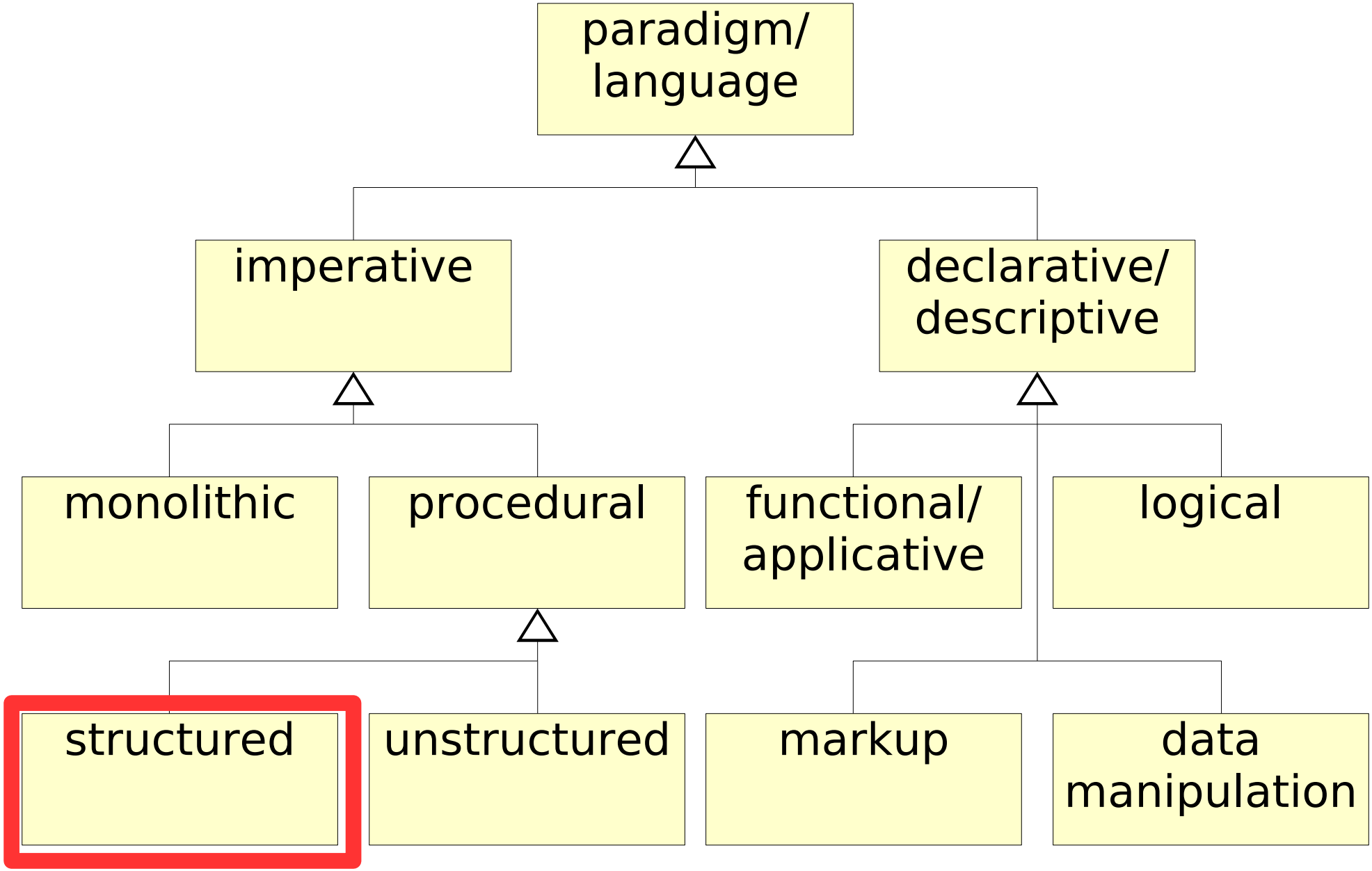
<> ungleich => !=

Universal Language

- Standalone on Desktop
- Web Application
- Mobile App
- Microdevice
- Serverside Technologies
- Big Data



https://www.heise.de/developer/artikel/Java-als-universelle-Programmiersprache-3742037.html?seite=all


paradigm/
language
imperative
declarative/
descriptive
monolithic
procedural
functional/
applicative
logical
structured
unstructured
markup
data
manipulation

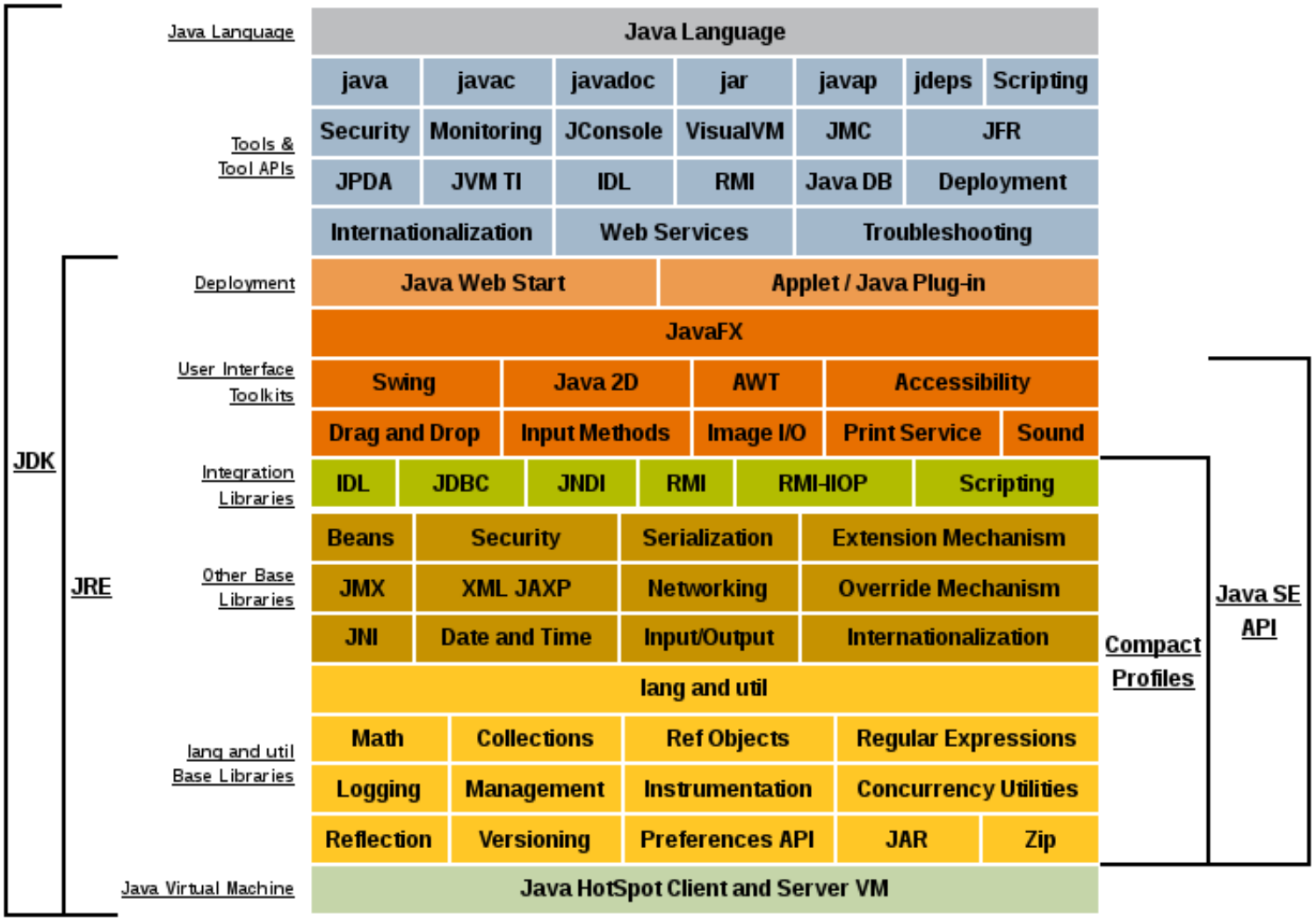
			Machine Language			
			Assembler			
1950		Fortran				
	APL	Algol		JOVIAL	Lisp	COBOL
1960	Simula	Basic	CPL	CORAL	Snobol	PL/I
			BCPL		ISWIM	MUMPS
		Smalltalk	B	Forth	Logo	
1970	Pascal		C	sh	Icon	Prolog
	Modula	MS Basic	CLU	Scheme	SASL	ML
	Ada	Mesa	awk	csh	sed	Rex
1980	B	Cedar	C w Classes	PostScript	Miranda	KRC
	Sharp APL	ObjectiveC	C++	Concurrent C	CommonLisp	SML
	ObjectPascal	Eiffel	nawk	OO Forth	Object Logo	
1990	Oberon	Self	Perl	Tcl/Tk	Haskell	Caml
	J	ABC	ANSI C	bash	Clos	
	Python	Visual Basic	Sather	Oak		NetRexx
2000	A+	Ruby	Cmm	ksh		M
	Delphi	PHP	JavaScript	Java		Open M
	K		JScript	ECMAScript	Objectv Caml	OO COBOL
			C#	Java 2	O Caml	

Java SE Versions

Version	Codename	Veröffentlichung
JDK 1.1.4	Sparkler	12. September 1997
JDK 1.1.5	Pumpkin	3. Dezember 1997
JDK 1.1.6	Abigail	24. April 1998
JDK 1.1.7	Brutus	28. September 1998
JDK 1.1.8	Chelsea	8. April 1999
J2SE 1.2	Playground	4. Dezember 1998
J2SE 1.2.1	(keiner)	30. März 1999
J2SE 1.2.2	Cricket	8. Juli 1999
J2SE 1.3	Kestrel	8. Mai 2000
J2SE 1.3.1	Ladybird	17. Mai 2001
J2SE 1.4.0	Merlin	13. Februar 2002
J2SE 1.4.1	Hopper	16. September 2002
J2SE 1.4.2	Mantis	26. Juni 2003
J2SE 5.0 (1.5.0)	Tiger	29. September 2004
JSE 6.0	(Mustang)[5]	11. Dezember 2006
JSE 7.0	(Dolphin)[5]	28. Juli 2011[6]
JSE 8.0	–	18. März 2014[7]
JSE 9.0	-	23. März 2017[8][9]

https://de.wikipedia.org/wiki/Java-Technologie

Java Platform Standard Edition (Java SE)



http://docs.oracle.com/javase/8/docs/index.html

Java Platform Runtime Environment (JRE)

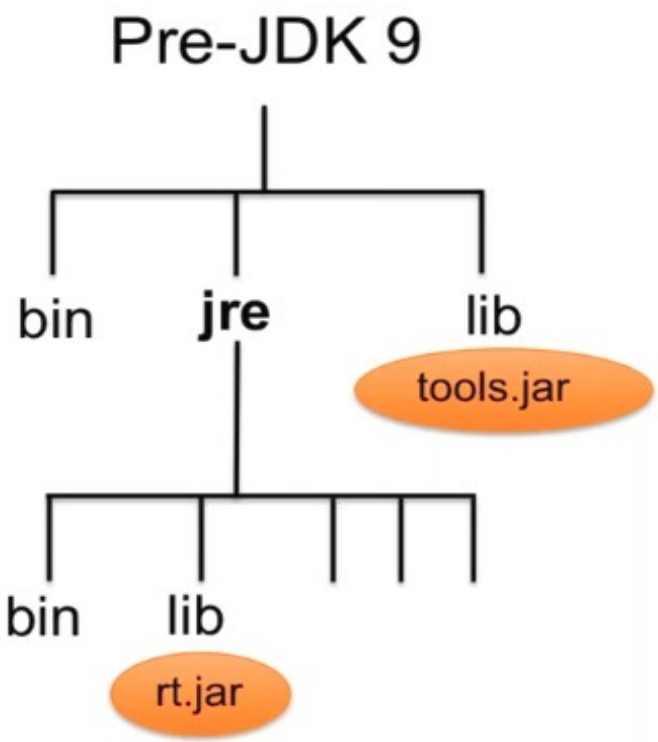
- Java Card
- Java Micro Edition (Java ME)
- Java Standard Edition (Java SE)
- Java Enterprise Edition (Java EE)

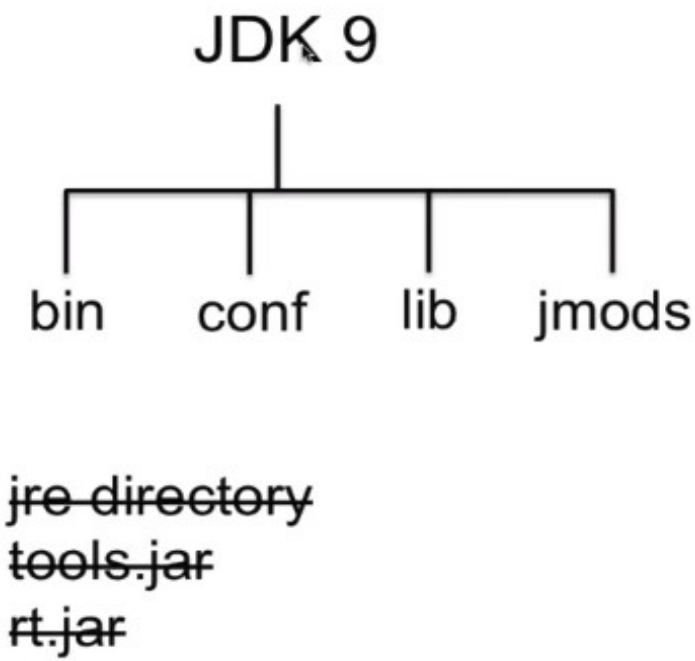
… sorted by size

Java Development Kit (JDK)

- Java Compiler (javac)
- Documentation tool (javadoc)
- Archiver (jar)
- Signing tool (jarsigner)
- Java Plug-in HTML Converter (htmlconverter)
- Applet Viewer (appletviewer), a simple browser

File Structure

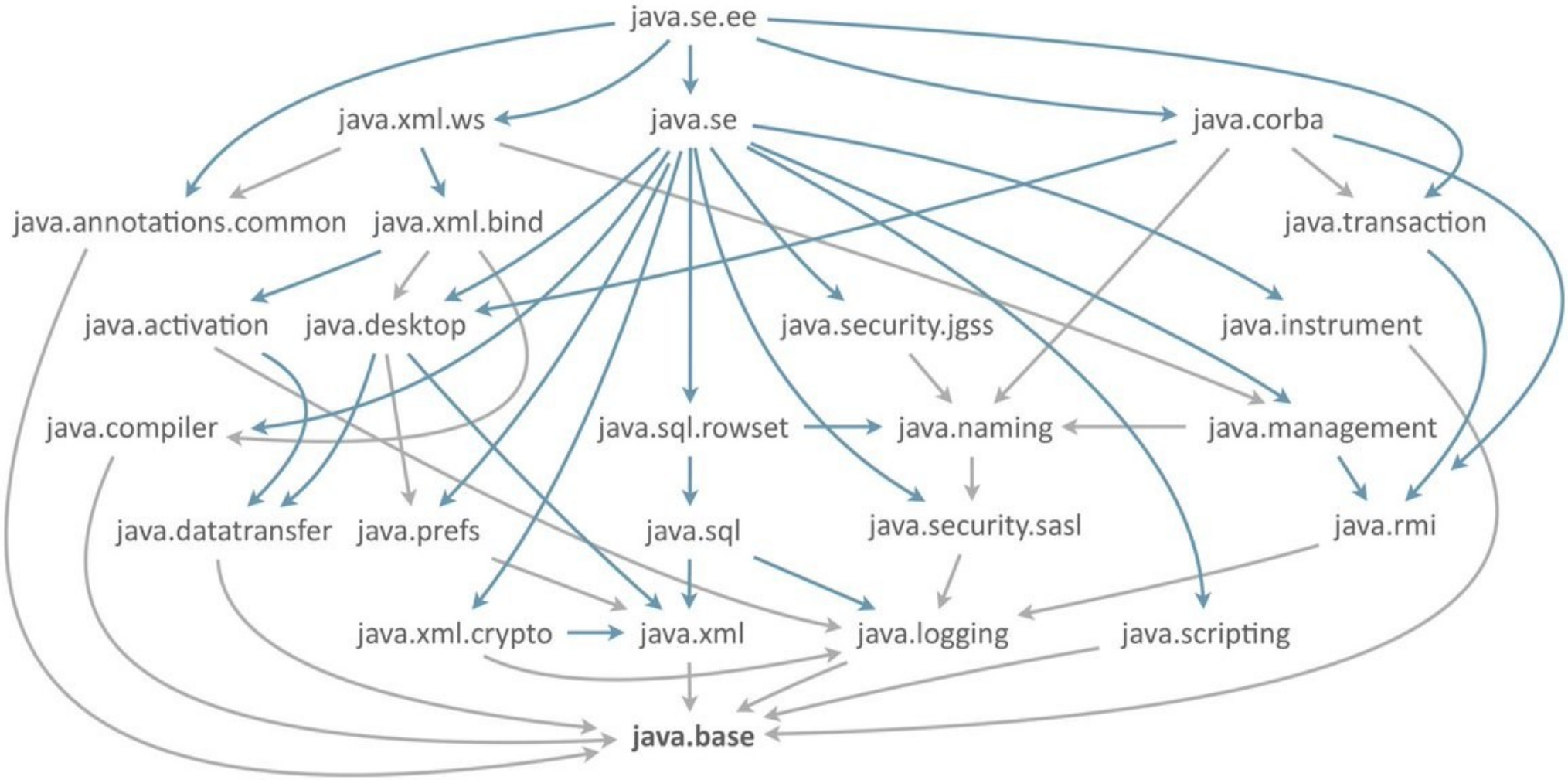




No more JRE, only JDK

David Shevchenko: Life after Java 8. 2018 https://blog.mimacom.com/life-after-java-8/

Modular System (Dependencies)



David Shevchenko: Life after Java 8. 2018 https://blog.mimacom.com/life-after-java-8/

Third Party Tools

Maven	3.5.0; compiler plugin 3.7.0
gradle	4.2.1
spring	5.0 4.3 - not first class support
spring boot	2.0
HIBERNATE	5.2
JUnit	5
Tomcat	8.x
jetty://	9.4
Netty	4.1.15
WildFly	11

David Shevchenko: Life after Java 8. 2018 https://blog.mimacom.com/life-after-java-8/

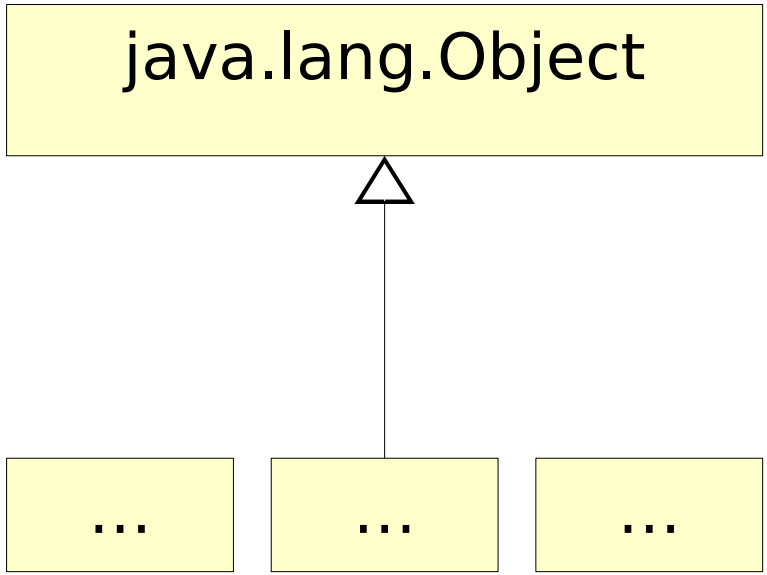
Java Virtual Machine (VM)

- Java Security Model
- Garbage Collector (GC)

Java Class Library

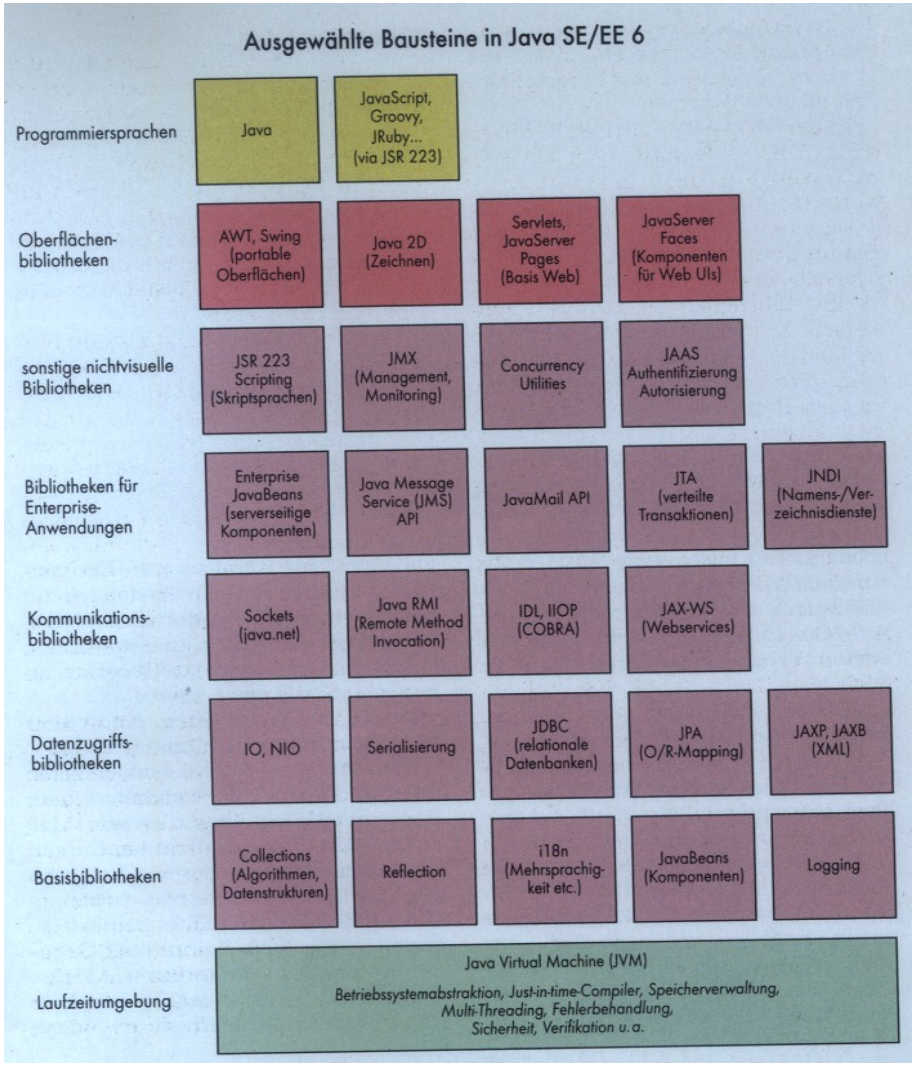
Version	Pakete	Klassen, Enums und Interfaces
JDK 1.0	8	212
JDK 1.1	23	504
JDK 1.2	59	1520
JDK 1.3	76	1842
JDK 1.4	135	2991
J2SE 5.0	166	3279
JSE 6.0	203	3793
JSE 7.0	209	4024
JSE 8.0	217	4240
JSE 9.0	n.a.	n.a.

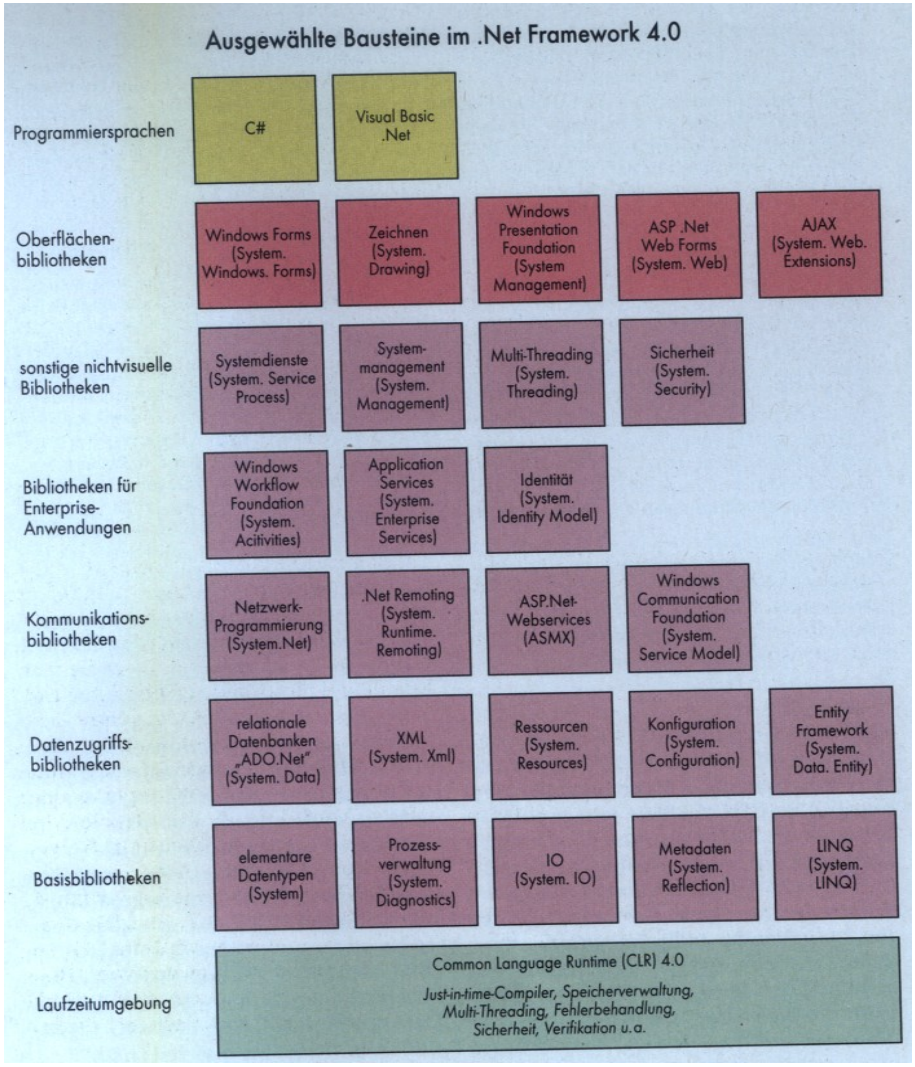
https://de.wikipedia.org/wiki/Java-Technologie



All later courses will deal with parts of the Java Class Library

Java SE/EE versus .NET



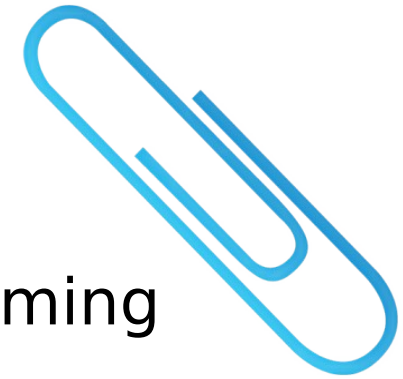


Heise Verlag. ix Magazin. April 2010

JDK Documentation and API Specification

- https://docs.oracle.com/en/java/javase/15/
- https://docs.oracle.com/en/java/javase/15/docs/api/

Summary



- type-based general purpose system programming
- multi-paradigm: structured, procedural, object-oriented
- class library with super class java.lang.Object
- modular system (before: SE, EE, ME)
- API documentation

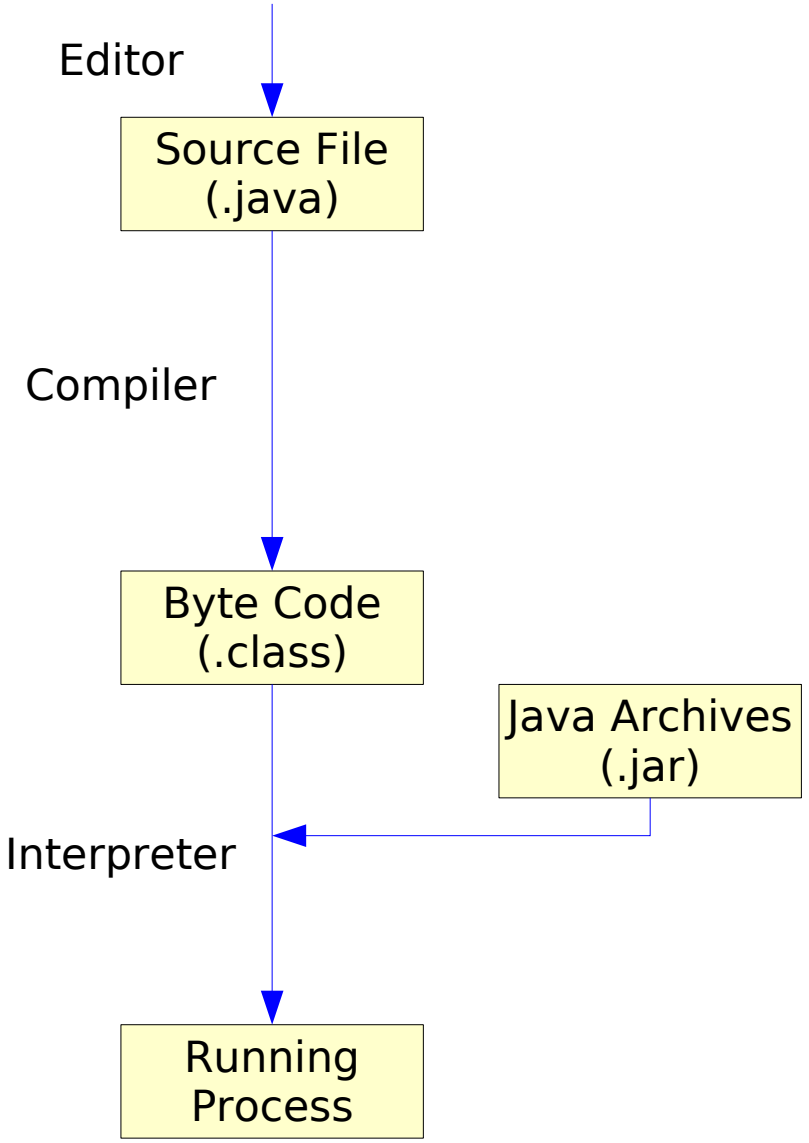
JDK

```
JRE
java
javac
javadoc
jar
```

Contents

Motivation

Development Tools

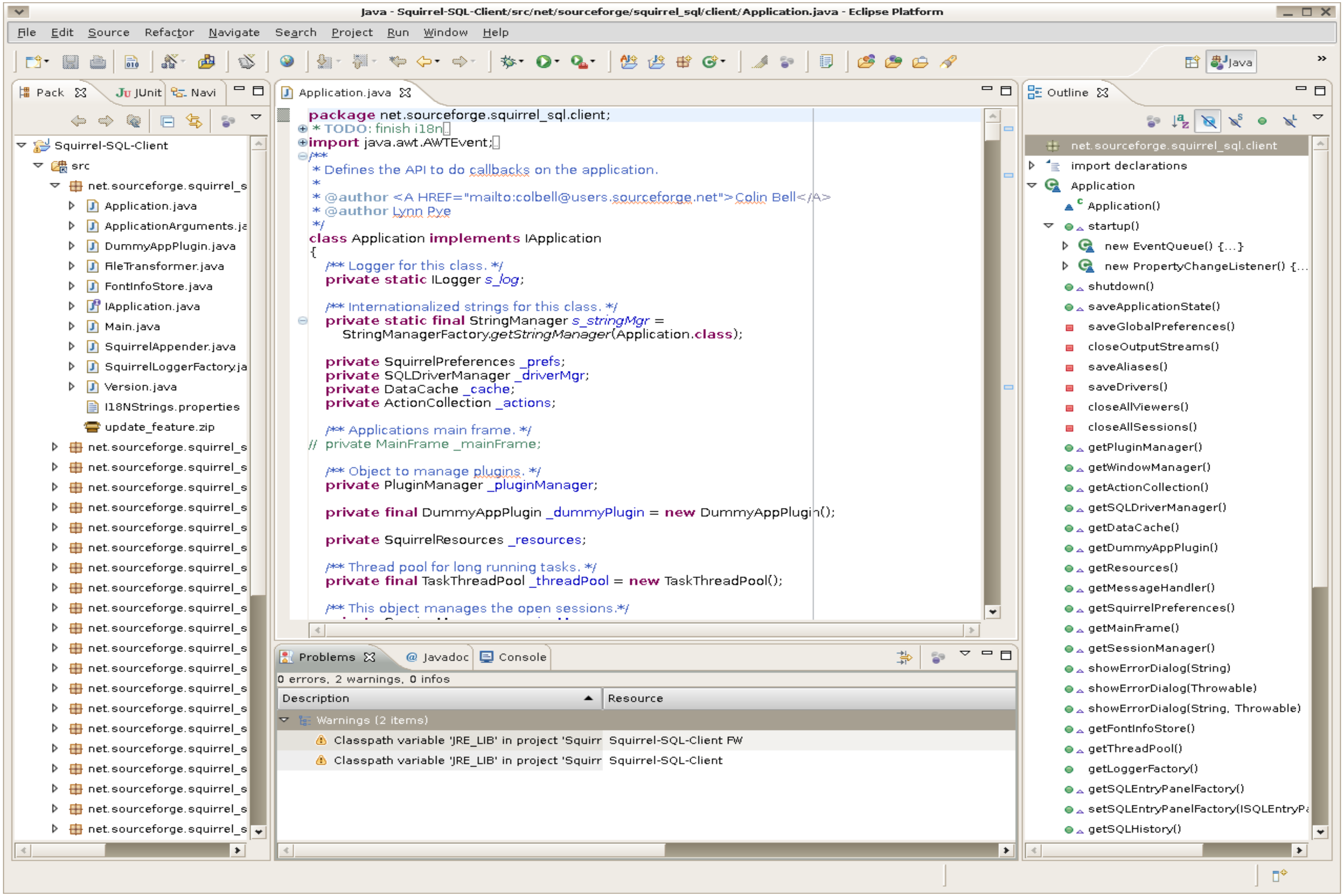


Hybrid language:
compiled + interpreted

Integrated Development Environment (IDE)

- Text-Editor (e.g. jEdit) and text terminal (CLI)
- Eclipse (successor of IBM Visual Age for Java)
- NetBeans (open source version of Sun Java Studio)
- IntelliJ IDEA (JetBrains)

Integrated Development Environment (IDE)

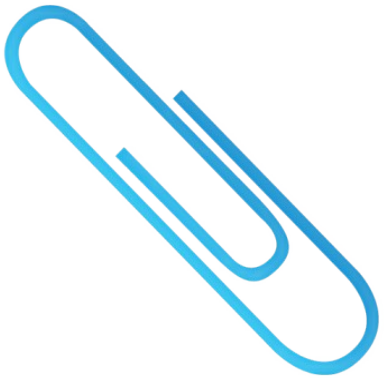


```
package net.sourceforge.squirrel_sql.client;
* TODO: finish i18n
import java.awt.AWTEvent;
/**
 * Defines the API to do callbacks on the application.
 *
 * @author <A HREF="mailto:colbell@users.sourceforge.net">Colin Bell</A>
 * @author Lynn Pye
 */
class Application implements IApplication
{
    /** Logger for this class. */
    private static ILogger s_log;

    /** Internationalized strings for this class. */
    private static final StringManager s_stringMgr =
        StringManagerFactory.getStringManager(Application.class);

    private SquirrelPreferences _prefs;
    private SQLDriverManager _driverMgr;
    private DataCache _cache;
    private ActionCollection _actions;

    /** Applications main frame. */
//  private MainFrame _mainFrame;

    /** Object to manage plugins. */
    private PluginManager _pluginManager;

    private final DummyAppPlugin _dummyPlugin = new DummyAppPlugin();

    private SquirrelResources _resources;

    /** Thread pool for long running tasks. */
    private final TaskThreadPool _threadPool = new TaskThreadPool();

    /** This object manages the open sessions.*/
```

0 errors, 2 warnings, 0 infos

Description	Resource
Warnings (2 items)	
⚠ Classpath variable 'JRE_LIB' in project 'Squirr	Squirrel-SQL-Client FW
⚠ Classpath variable 'JRE_LIB' in project 'Squirr	Squirrel-SQL-Client

Summary



- hybrid language: compiled and interpreted
- IDE: combines many tools

Tools

```
editor
compiler
interpreter
debugger
```

Contents

Motivation

Semiotics - Language Definition

- Semantics
- Syntactics (Syntax)
- Pragmatics

- Vocabulary

Reserved Keywords

nicht für Variablen benutzbar

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

Identifier

mami

kulliReimtSichAufUlli

Teilwörter mit großem Buchstaben beginnen (Camel-Case)

IchWeißIchMussAndréAnrufen

Unicode zeichen

RAPHAEL_IST_LIEB

2und2macht4

— mit Ziffer beginnen ist verboten

class

hose_gewaschen

hurtig!

Naming Conventions

- Attribute / Member / Instance- and Class Variable
- Class / Compound Type / Container / Array
- Method / Procedure / Function / Routine / Algorithm

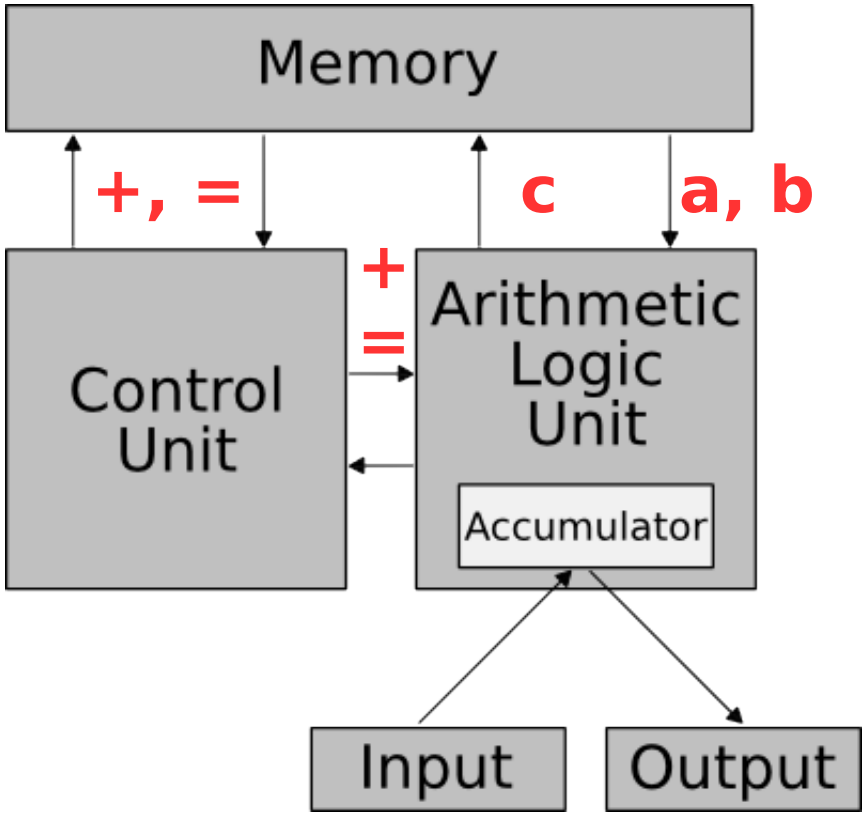
Comment

// Single line comment

/* Block comment (not nestable) */

/** Special block comment interpreted by javadoc */

John von Neumann Computer Architecture



int a = 1, b = 2, c;
c = a + b;

https://de.wikibooks.org/wiki/Computergeschichte:_1900_bis_heute

State and Logic

c = a + b

c, a, b, +, =

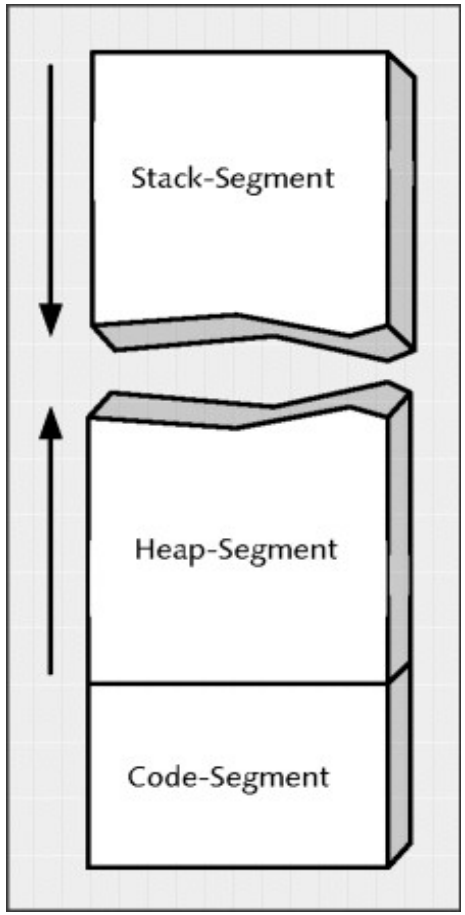
Summenoperation

State
Quality or Quantity
Space
Symbol or Number
Variable
Operand
Argument
Parametre
Attribute

Logic
Change
Time
Algorithm
Operator
Operation
Function
Procedure
Method

Memory Segments

- Text/Code: instructions, literals, static
- Data/Heap: working storage
- Stack: programme stack



Jürgen Wolf: C von A bis Z.
Rheinwerk (Galileo Computing), 2009

See also:
http://www.gnu.org/software/libc/manual/html_mono/libc.html#Memory-Concepts

Variable

- parametre
- local variable
- class variable (static field)
- instance variable (non-static field)

Parameter

```
/**
 * A method which gets handed over three parameters and returns one value.
 */
int go(int p0, int p1, int p2) {
    ↳ Methoden/Fkt.-Namen
    …
}
```

Local Variable

```
int go() {

    // The maximum iterations as temporary variable.
    int max = 10;
    // The loop counter as temporary variable.
    int j = 0;

    while (j < max) {

        System.out.println(j);

        j++;
    }

    return j;
}
```

Class Variable (Static Field)

```
class Bicycle {

    // Class variable / attribute.
    static int gearCount;
}
```

Instance Variable (Non-Static Field)

```
class Bicycle {

    // Instance variable / attribute.
    int speed;
}
```

Variable's Terminology

```
// Declaration. Bekanntmachung der Variable mit Typ + Variablenname
int i;
double d;

// Initialisation. Anfangswert zuweisen
int i = 0;
String s = "";

// Instantiation of compound type.
String s = new String("test");

// Method signature / header.
void method();

// Method implementation / definition.
void method() {
    ...
}
```

Bit & Byte

- Binary digit (Bit)
  Binäre Zähleinheit
- Crumb
  2 Bit kombiniert
- Nibble
  4
- Byte
  8
- Word
  2 Byte
- Double Word (DWord)

Data Type

Primitive

Zsh-Gesetz
Compound

Boolean

- boolean

Number

Zeichen
Character

- char

Array

Class

(Object)

ganzzahl
Integral

- byte
- short
- int
- long

Kommazahlen
Floating Point

- float
- double

Data Type Value Range

Signed Data Types bei Java

Data Type	Length Byte	Default Value	Range
boolean	1	false	true or false
byte	1	0	–2^7 ... 2^7 – 1 (–128 ... 127)
short	2	0	–2^15 ... 2^15 – 1 (–32768 ... 32767)
int	4	0	–2^31 ... 2^31 – 1 (–2147483648 ... 2147483647)
long	8	0L	–2^63 ... 2^63 – 1 (–9223372036854775808 ... 9223372036854775807)
float	4	0.0f	1,40239846E-45f ... 3,40282347E+38f
double	8	0.0d	4,9406564584124654 4E-324 ... 1,79769313486231570E+308
char	2	'\u0000'	16-Bit Unicode Zeichen (0x0000 ... 0xFFFF)
String / Object	-	null	-

Floating Point Number

9,8 7 6 5 4 3 2 1 (große Genauigkeit)
9 8 7 6 5 4 3 2,1 (großer Wertebereich)
9 8 7 6 5,4 3 2 1 (Mischung?)

- Integer: simple representation, fast calculation
- Fixed-point number: historic, fast, limited value range
- Floating-point: various representations, mostly IEEE 754
- Floating Point Unit (FPU): part of modern CPU
- Problem: float representations imprecise approximation
- Solution: class BigDecimal in Java, very slow (x 100)

https://de.wikipedia.org/wiki/Festkommazahl
https://de.wikipedia.org/wiki/Gleitkommazahl
http://www.elektronik-kompendium.de/sites/dig/1807231.htm
https://www.holisticon.de/2013/08/korrekte-berechnungen-mit-praezision-in-java/

Expression of Literal Types

```
// Literal
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

```
// Integral Type
int decVal = 26;
int octVal = 032;
int hexVal = 0x1a;
```

```
// Character
'c'
'\u0108' (capital Ĉ with circumflex)

// String
"test_string"
"S\u00ED se\u00F1or" (Sí Señor in Spanish)
```

```
// Floating Point Number
double d1 = 123.4;
double d2 = 1.234e2;
float f1 = 123.4f;
```

```
// Non-existing object type
Object o = null;
```

Escape Sequences for Character and String Literals

\b - backspace

\t - tab

\n - line feed

\f - form feed

\r - carriage return

\" - double quote

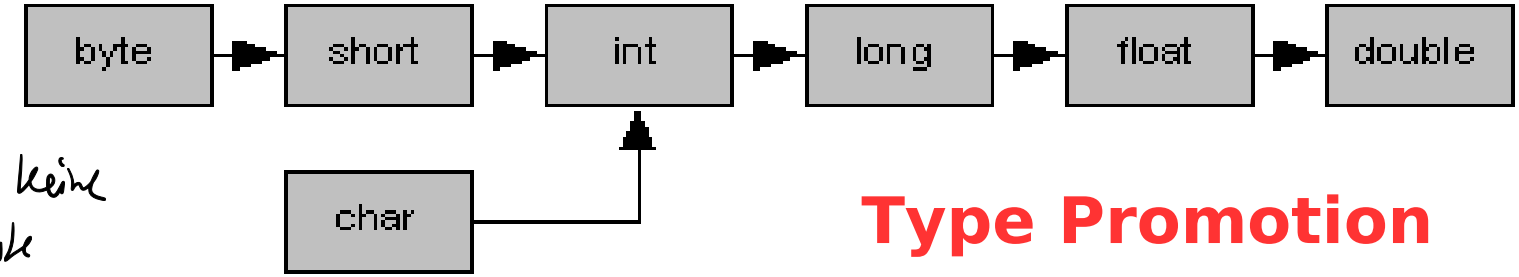
\' - single quote

\\ - backslash

Type Cast / Conversion

Typen überführen

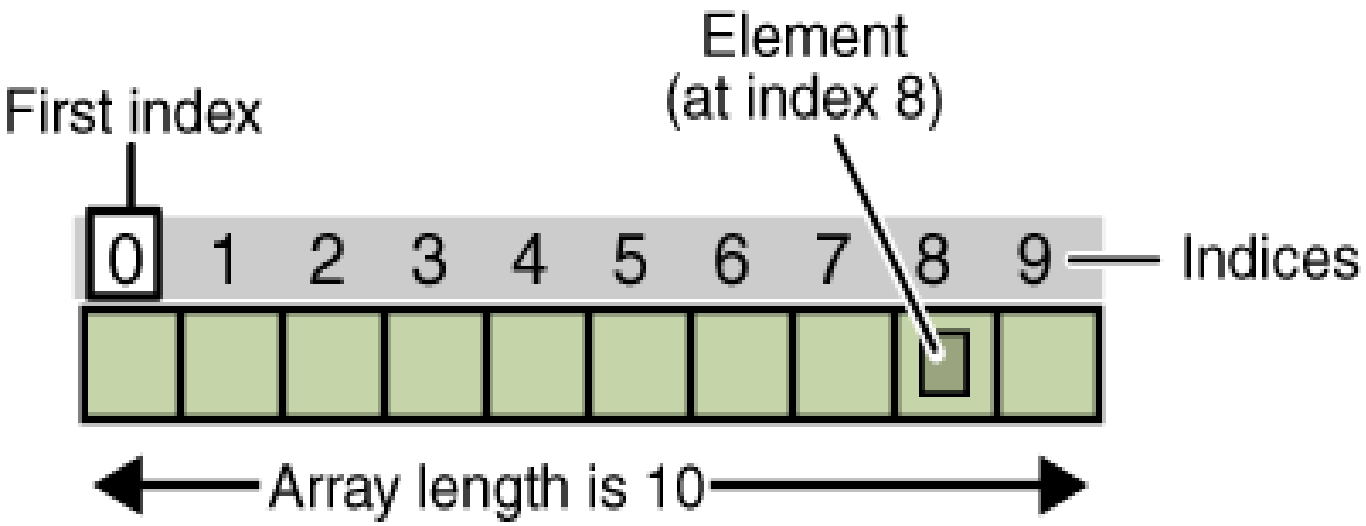
```
// Explicit type cast (without information loss, since byte covers numbers to 127.
int i = 100;
byte b = (byte) i;

// Implicit type cast (since a value of type int gets assigned to a double variable).
int j = 12;
double d = j;

// Implicit type cast as source of errors.
int x = 9;
// The z below is not 4.5, but 4.0, as in the equivalent writing with parentheses:
// double z = (x / 2);
// Since x is an int, the division returns an int as well.
// Only after having calculated and returned the int, it gets converted to double.
double z = x / 2;
// An equivalent example (using type int for result of division) would be:
int y = x / 2;
z = y;
// The workaround for this behaviour is to write the divisor as double,
// so that the result gets calculated and returned as double, too.
int u = 9;
double w = u / 2.0;
```



Schnittmenge keine Verluste

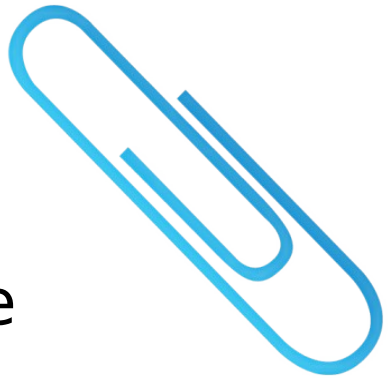
Type Promotion

Array



https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Summary



- variable: reserved memory containing a value
- kinds: parametre, local, class field (static/global), instance
- data type: define size of variable and possible operations
- kinds: primitive, compound (class)
- cast: conversion to another data type
- array: many values of one single type

Keywords

```
int
double
boolean
String
```

Contents

Motivation

State and Logic

c = a + b

State
Quality or Quantity
Space
Symbol or Number
Variable
Operand
Argument
Parameter
Attribute

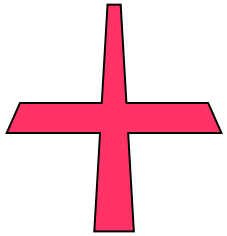
Logic
Change
Time
Algorithm
Operator
Operation
Function
Procedure
Method

Operator	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment and compound assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Higher Precedence

4 + 8 * 2 = 4 + (8 * 2)

Operator



- Arithmetic
- Comparison
- Boolean Logic
- Assignment
- Bit Manipulation
- Access
- Memory Management
- Miscellaneous

Arithmetic Operators

+, -	- Addition, Subtraction
*	- Multiplication
/, %	- Division, Modulo Division
+, -	- Sign (unary)
++, --	- Increment, Decrement

↳ Wert hinzuaddieren/subtrahieren um 1

Comparison Operators

==	- equal
!=	- unequal
<	- smaller
<=	- smaller or equal
>	- greater
>=	- greater or equal

Boolean Logic Operators

!	- NOT
&&	- AND
||	- OR
	- XOR (ex klusive Or → bei verschiedlichen

1 0
0 1

Assignment Operators

= - Simple assignment

op= - Composed assignment with "op" being either:

- a binary arithmetic operator
- a binary bit operator

Bit Manipulation Operators

~ - NOT (one's complement)

&, | - AND, OR

^ - Exclusive OR

→ schieben

<<, >> - Shift left (multiplication)/ right (division)

```
// Variable with 8 Bit. Java knows only signed types. Value range: -128...127
byte i;

// Type casts are necessary, since bit operators return type int.
i = (byte) 1;                  // 0000 0001 → 2^0                    → 2^3      1
i = (byte) (i << 3);           // 0000 0001 << 3          = 0000 1000        8
i = (byte) (i >> 2);           // 0000 1000 >> 2          = 0000 0010 → 2^1  2
i = (byte) (i | 5);            // 0000 0010 | 0000 0101   = 0000 0111        7
i = (byte) (i & 3);            // 0000 0111 & 0000 0011   = 0000 0011        3
i = (byte) (i ^ 5);            // 0000 0011 ^ 0000 0101   = 0000 0110        6
i = (byte) ~i;                 // ~ 0000 0110             = 1111 1001        -7
// last result is NOT 249, but rather -7 which is the two's complement due to leading 1
```

Access Operators

[] - Index

. - object.element

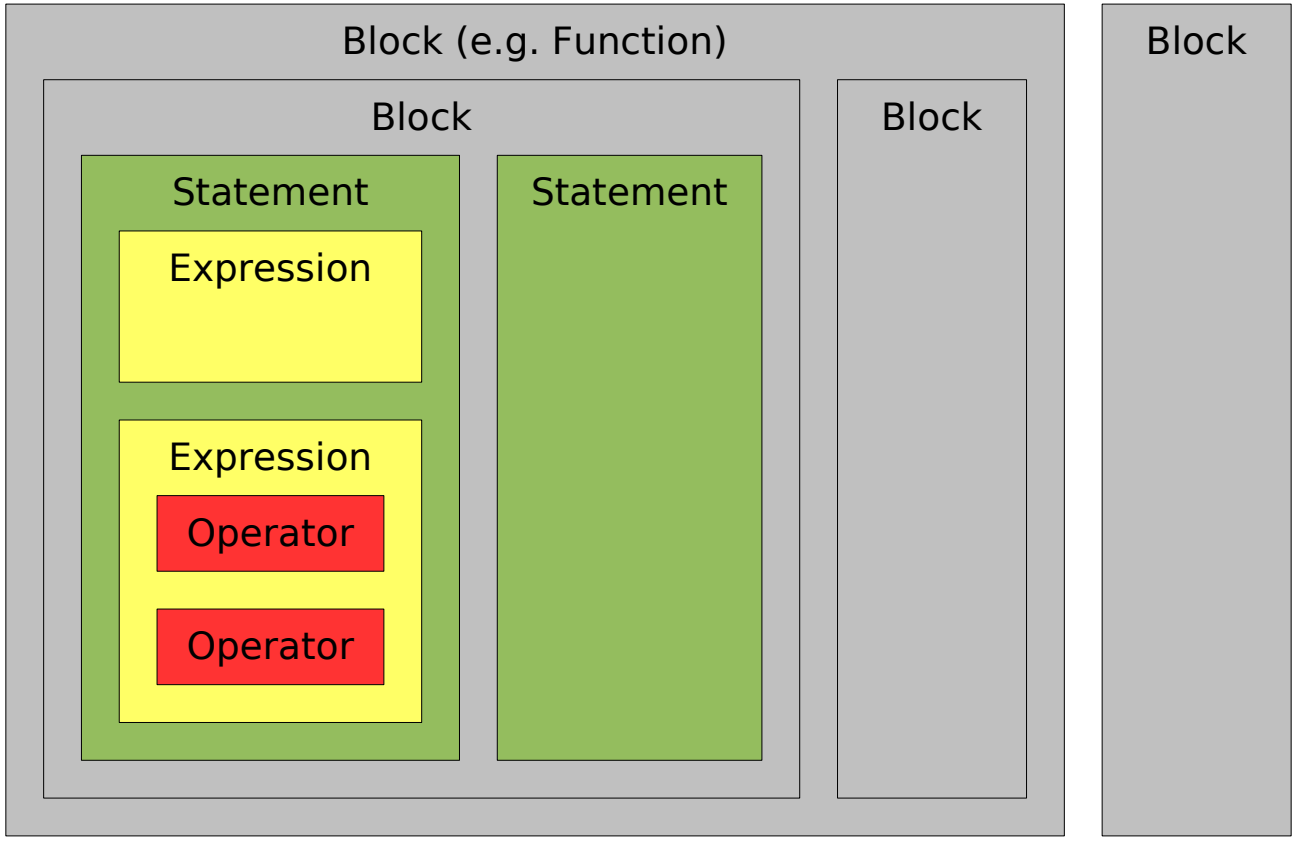
Memory Management Operators

new - Create object (instance) dynamically

Miscellaneous Operators

?:	- conditional selection (if-then-else)
,	- comma (for enumerations)
name()	- function call
instanceof	- verify if object is an instance of a class

Source Code Grouping and Nesting



Expression

```
anArray[0] = 100
result = 1 + 2
value1 == value2
```

Be explicit and indicate with parentheses which operators should be evaluated first!

Statement

```
// Declaration statement.
double aValue;

// Assignment statement.
aValue = 8933.234;

// Initialisation statement.
double aValue = 8933.234;

// Increment statement.
aValue++;
```

Block

```
// Example using blocks
{ statement1; statement2; ... { statement3; ... } }
```

```
// Alternative writing using formatting
{
    statement1;
    statement2;
    ...

    {
        statement3;
        ...
    }
}
```

Summary

- operation: system of rules, performed on operands
- precedence: priority of operators
- operator: arithmetic, comparison, boolean, assignment, bit, access, memory management
- grouping: nested blocks

Keywords

=
+
==
!

Contents

Motivation

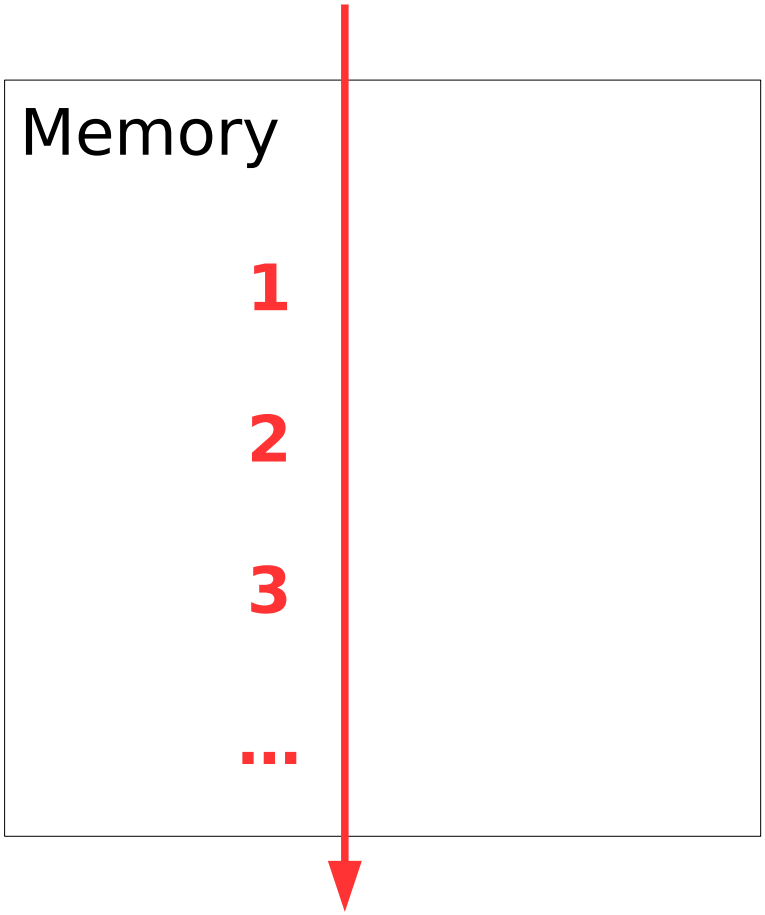
Jump / Goto

https://www.wikipedia.org/

Structured Programming

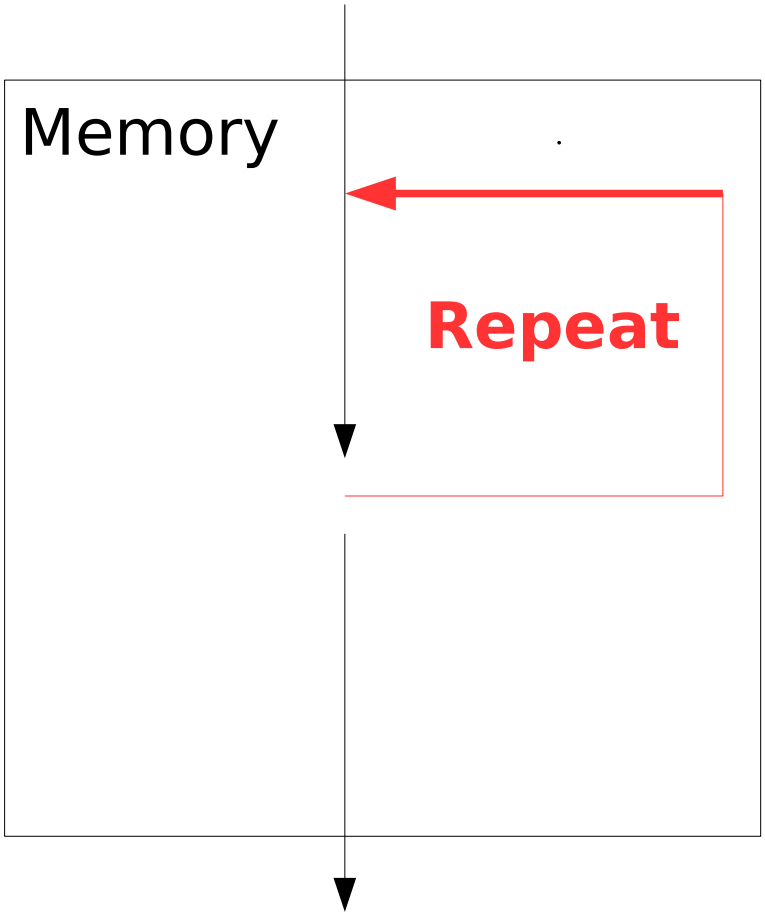
- Sequence
- Repetition
- Branching

Sequence / Concatenation




```
int a = 1;
int b = 2;
int c = a + b;
System.out.println("The sum is: " + c);
```

Repetition / Iteration / Loop



while / do ... while

```
int n = 100;
int sum = 0;
int i = 0;
while (i < n) {
    sum += i;
    i++;
}
System.out.println("The sum is: " + sum);
```

```
do {
    sum += i;
    i++;
} while (i < n)
```

for / for each

```
int n = 100;
int sum = 0;
for (int i = 0; i < n; i++) {
    sum += i;
}
System.out.println("The sum is: " + sum);
```

break / continue

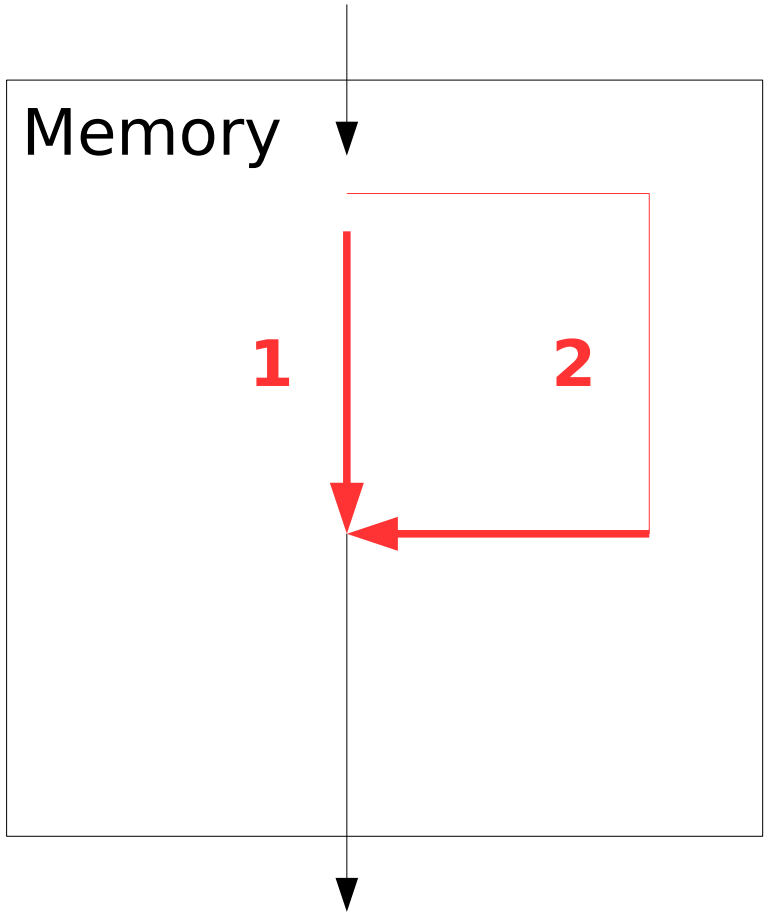
```
break: interrupt loop before having finished
continue: skip one cycle, jump to condition
```

Endless Loop

```
int i = 0;

while (true) {

    System.out.println("The endless loop is running ...");
    System.out.println("Current value of i: " + i);

    // The break condition may be situated inside the loop as well.
    if (i >= 100000) {

        break;
    }

    i++;
}
```

In shell, try pressing <ctrl> + <c> to exit

Branching / Selection / Choice / Condition



if ... else

```
int n = 8;
if ((n == 6) || (n == 8)) {
    System.out.println("Almost lucky");
} else if (n == 7) {
    System.out.println("Winner");
} else {
    System.out.println("Bad luck");
}
```

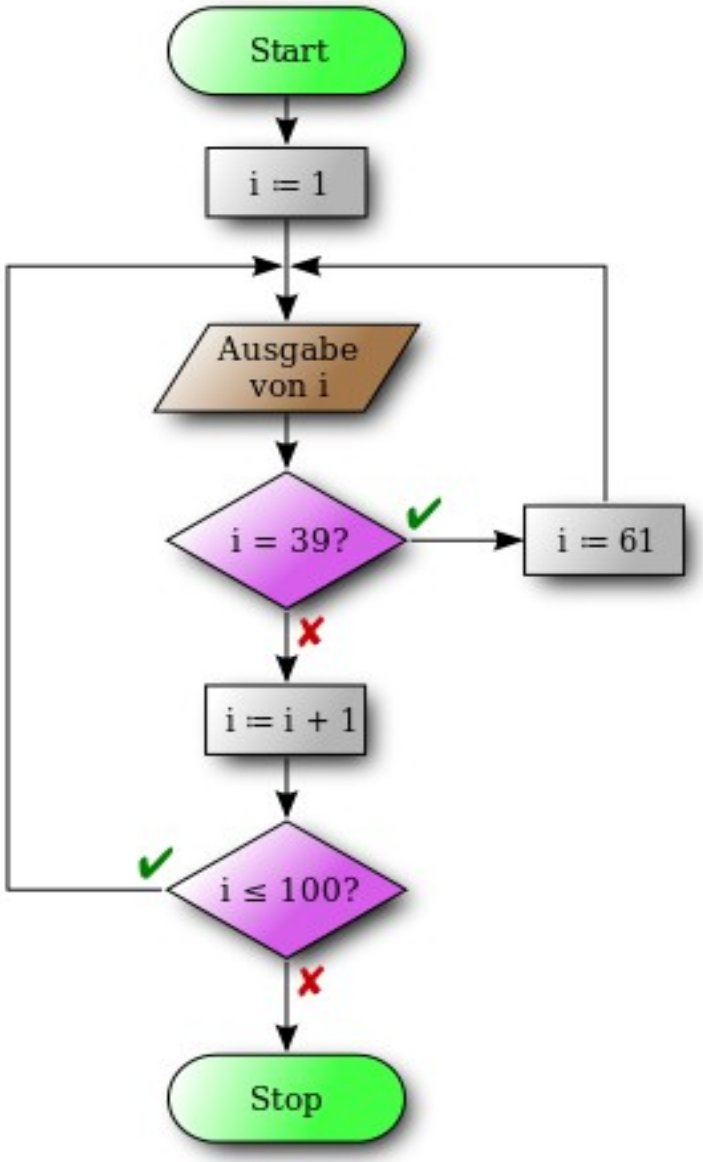
switch ... case

```
int n = 8;
switch (n) {
    case 6:
    case 8:
        System.out.println("Almost lucky");
        break;
    case 7:
        System.out.println("Winner");
        break;
    default:
        System.out.println("Bad luck");
}
```

Graphical Notation

- Programme Flow Chart
- Structure Chart
- Jackson Diagram
- Warnier/Orr Diagram

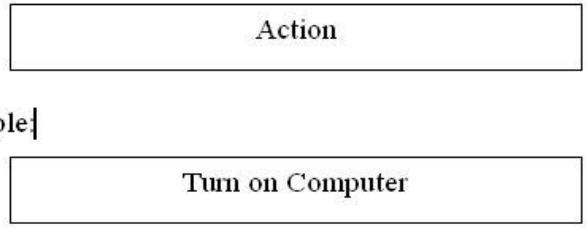
Programme Flow Chart



https://de.wikipedia.org/wiki/Programmablaufplan

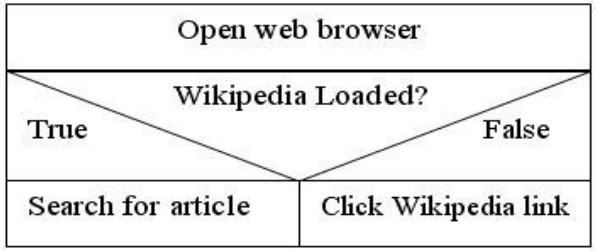
Structure Chart: Nassi-Shneiderman Diagram (NSD)

Standard Process Block:

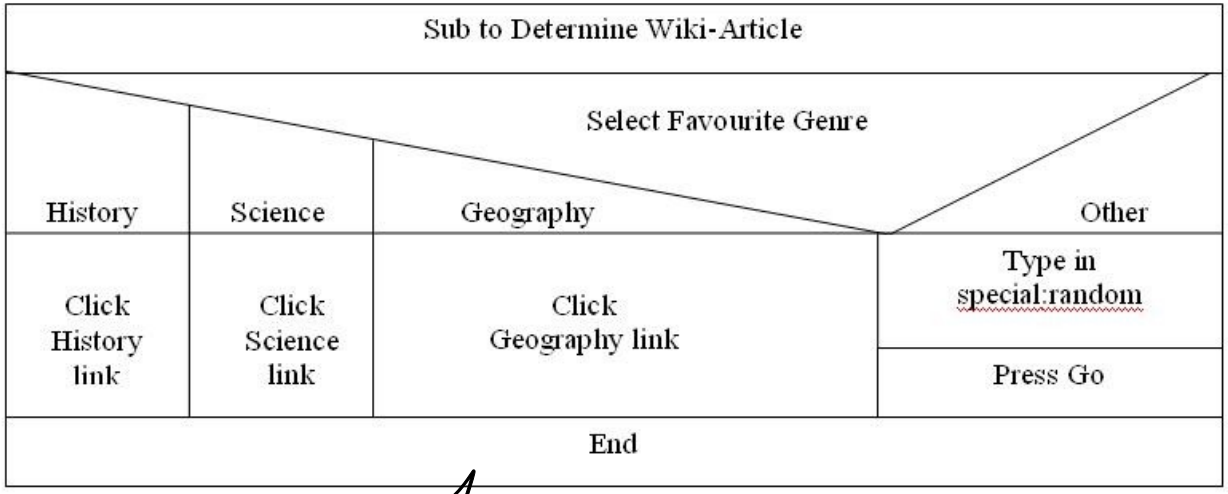


Example:

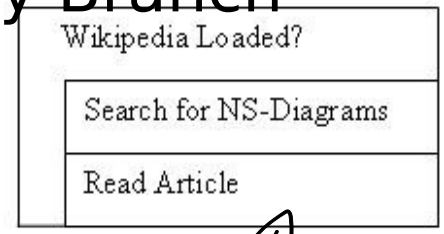
Process Block



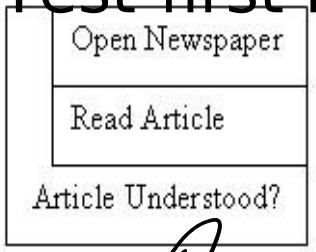
Two-way Branch



Multiple Branching



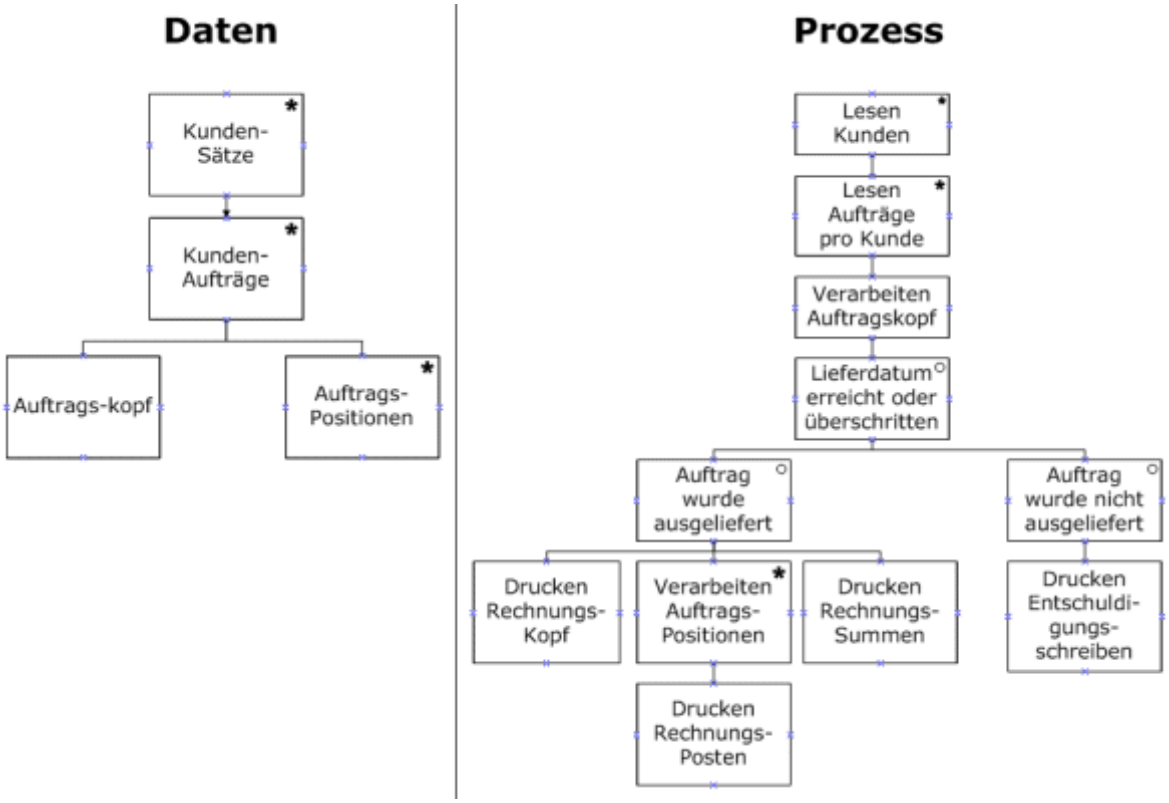
Test-first Loop



Test-last Loop

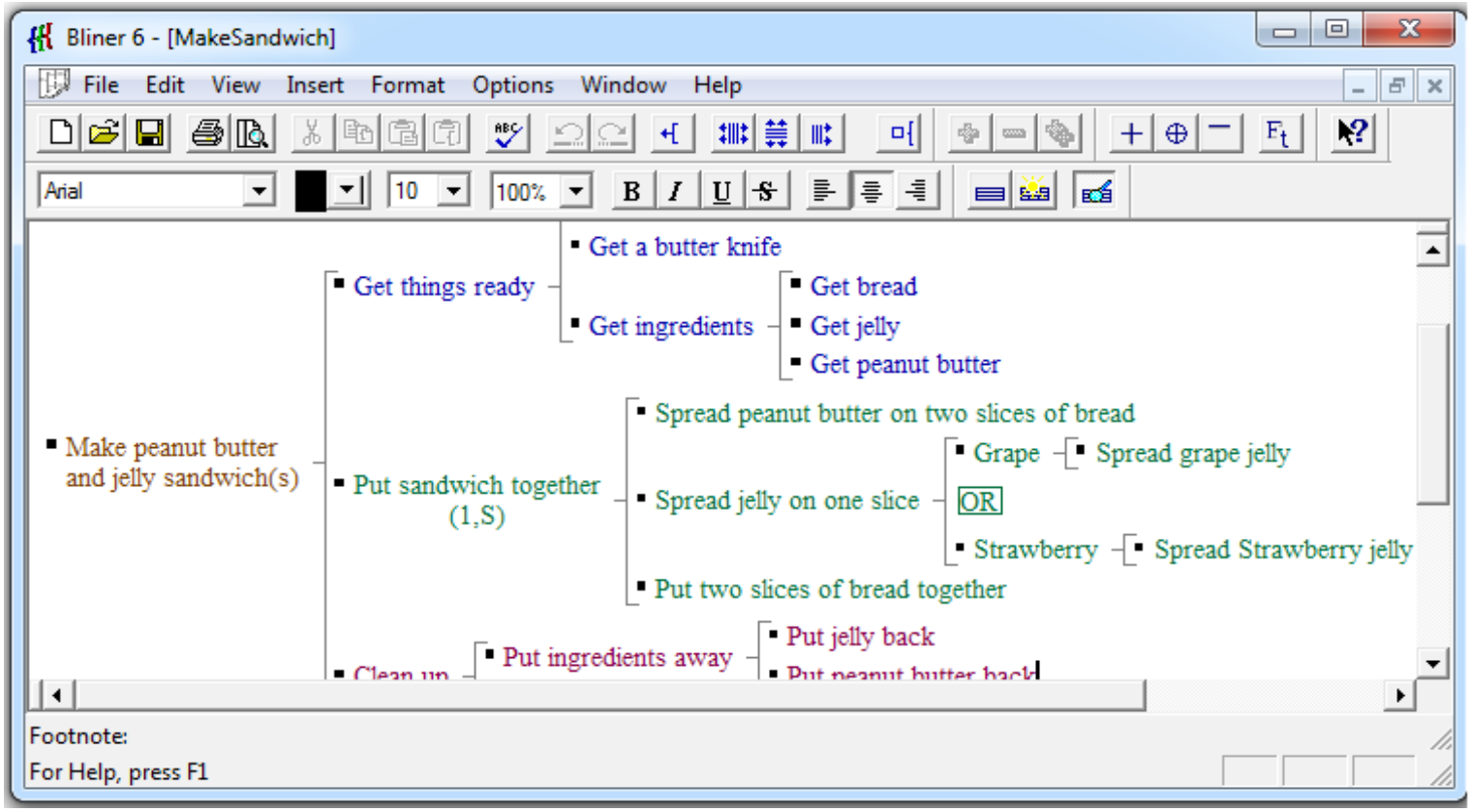
https://en.wikipedia.org/wiki/Nassi%E2%80%93Shneiderman_diagram

Jackson Structured Programming (JSP) Diagram



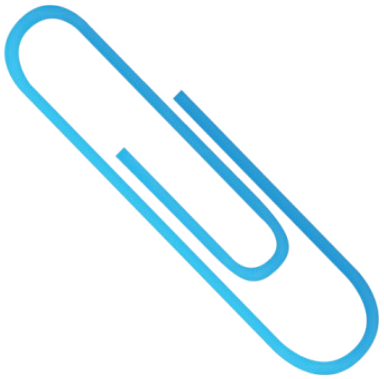
https://de.wikipedia.org/wiki/Jackson-Diagramm

Warnier/Orr Diagram Example "Make Sandwich"



http://www.davehigginsconsulting.com/pd03.htm
http://www.bliner.com/

Summary



- jump: goto
- concatenation: sequence
- repetition: (endless) loop, iteration
- selection: branching, choice, condition
- graphical notation

Keywords

```
while
do-while
for

break

if-else
switch-case
```

Contents

Motivation

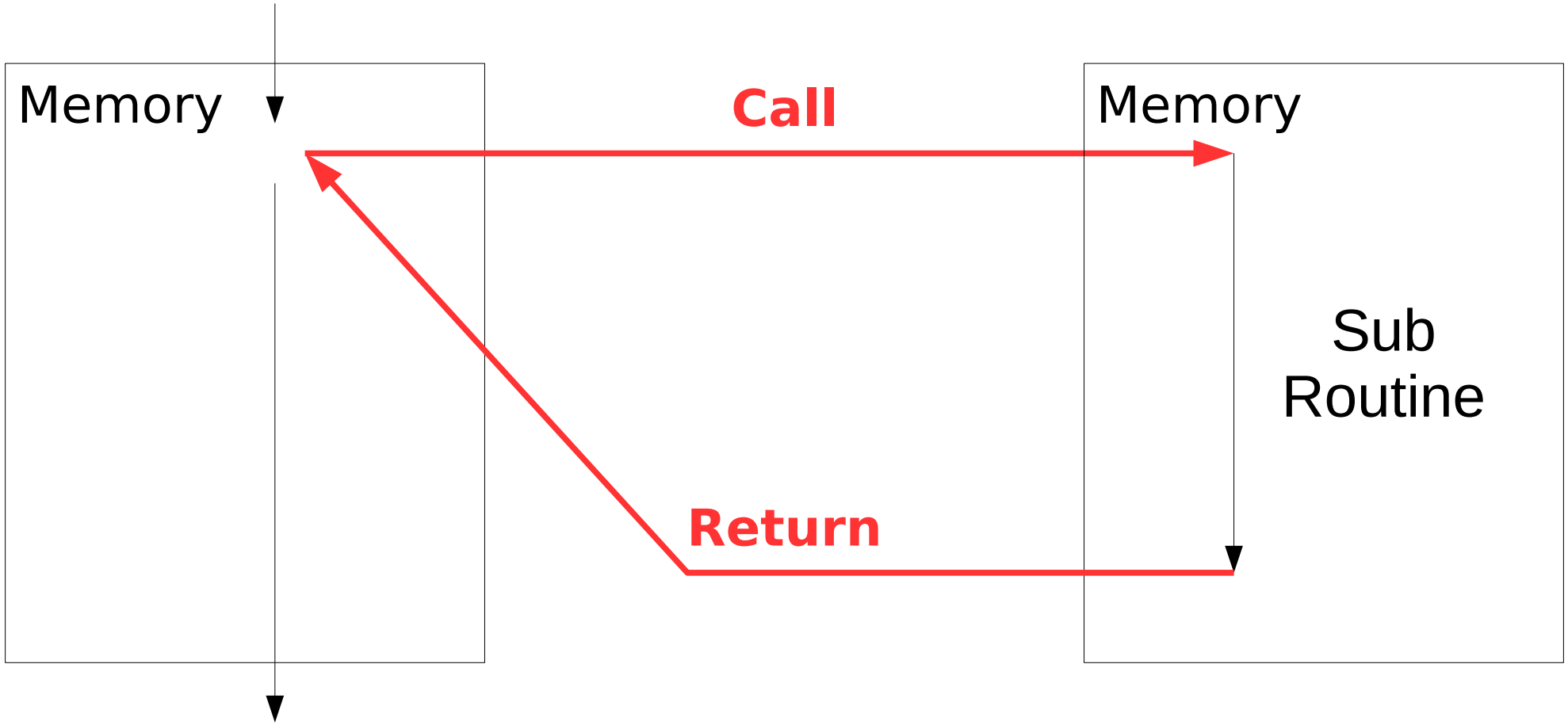
Reuse a programme

Procedural Programming

```
static int anExampleMethod(int i1, int i2, int i3) {

    // The result.
    int r = i1 + i2 + i3;

    return r;
}

public static void main(String[] args) {

    int value = anExampleMethod(1, 2, 5);

    System.out.println(value);
}
```

Procedure / Function / Routine / Method

Procedure Call




```
int a = 1;
int c = addSomeValue(a);
System.out.println("The result is: " + c);
// The result is: 3
```

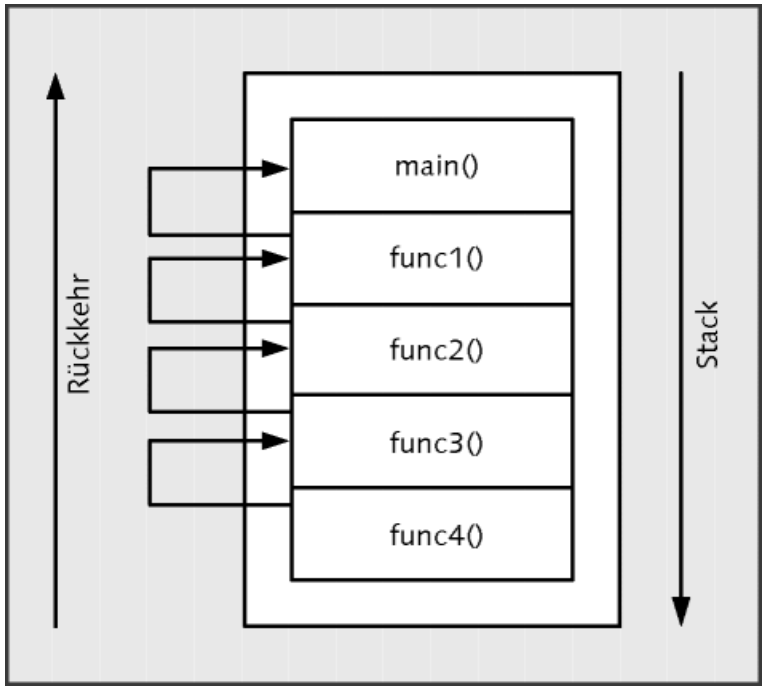
```
int addSomeValue(int x) {
    int y = 2;
    int r = x + y;
    return r;
}
```

Parameter, Local and Global Variable

```
class Launcher {

    static int aGlobalVariable = 5;

    public static void main(String[] aParameter) {

        int aLocalVariable = aGlobalVariable + aParameter[0];

        System.out.println("The sum is: " + aLocalVariable);
    }
}
```

Parameter Forwarding

```
class Launcher {

    static int incProcedure(int x) {

        x++;
        return x;
    }

    static int addProcedure(int a, int b) {

        // Forward parameter a to another procedure.
        a = incProcedure(a);
        int c = a + b;
        return c;
    }

    public static void main(String[] args) {

        int a = 1;
        int b = 2;
        int c = addProcedure(a, b);

        System.out.println("The result is: " + c);
        // The result is: 4
    }
}
```



Call Stack
Jürgen Wolf: C von A bis Z.
Rheinwerk (Galileo Computing), 2009

Pass (Call) by Value and by Reference

```
// A structure containing a field.
class Account {

    double balance = 1000;
}

public class Test {

    static void test(double v, Account a) {
        v = v + 100.0;
        a.balance = a.balance + 100.0;
    }

    public static void main(String[] args) {

        Account account = new Account();
        double value = 1000.0;

        System.out.println("Before: Value=" + value + " Balance=" + account.balance);
        Test.test(value, account);
        System.out.println("After: Value=" + value + " Balance=" + account.balance);
    }
}
```

After: Value=1000.0 Balance=1100.0

Pointer

- Variable, die eine Speicheradresse beinhaltet
- vor allem in maschinennahen Programmiersprachen

Reference

- constant pointer
- not changeable after creation
- automatically dereferenced at each access
- used for arguments and return values of functions

Recursion

```
public class Launcher {

    static int calculate(int n) {

        if (n < 10) {

            n = n + 1;

            n = calculate(n);
        }

        return n;
    }

    public static void main(String[] args) {

        // The number.
        int n = 0;

        n = calculate(n);

        System.out.println("The number is: " + n);
    }
}
```

Method Overloading

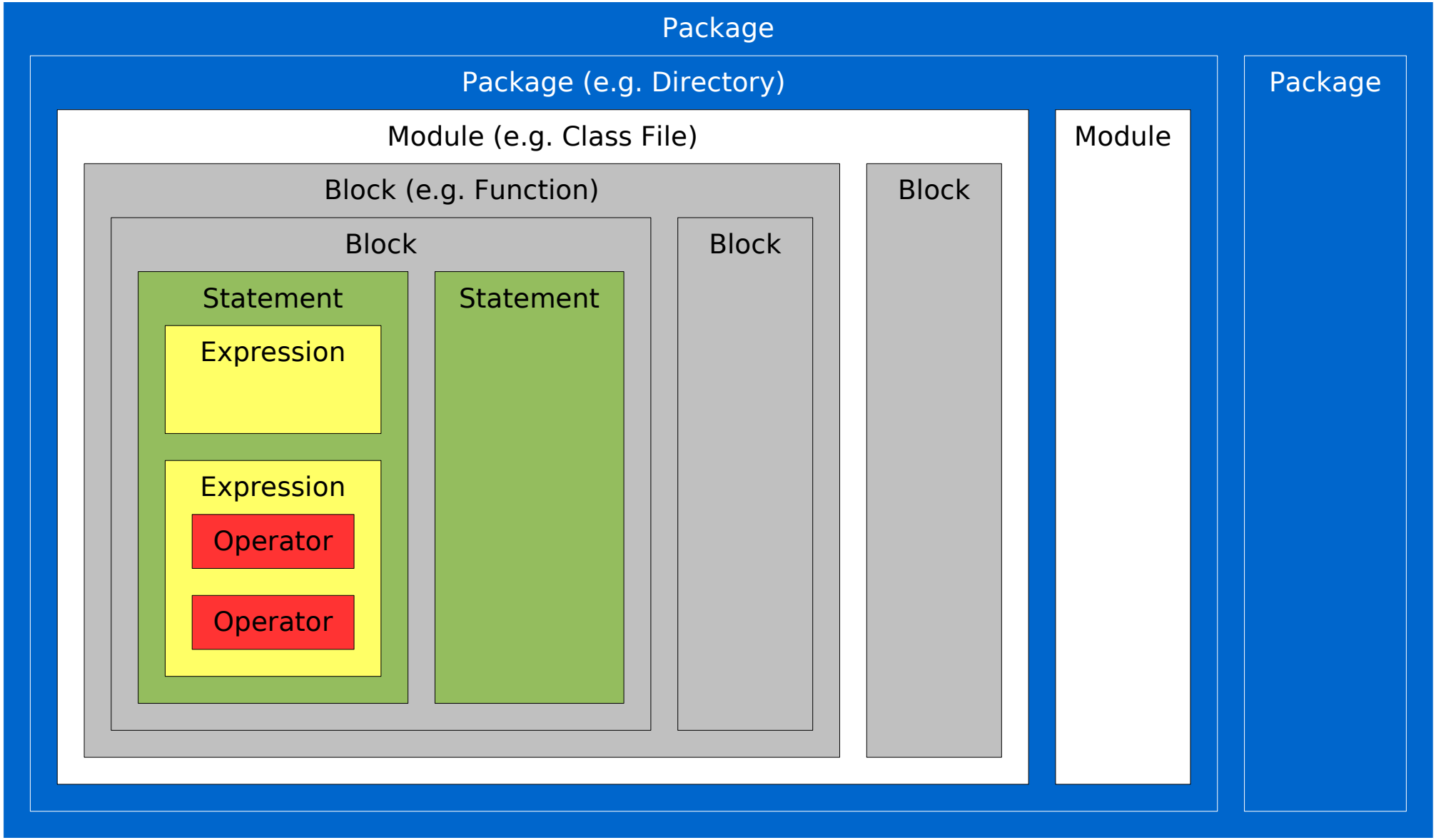
```
public class Launcher {

    static int test(int a) {

        return a * a;
    }

    static int test(int a, int b) {

        return a + b;
    }

    public static void main(String[] args) {

        int a = 2;
        int b = 3;
        int square = test(a);
        int sum = test(a, b);

        // The square is: 4
        System.out.println("The square is: " + square);

        // The sum is: 5
        System.out.println("The sum is: " + sum);
    }
}
```

Package

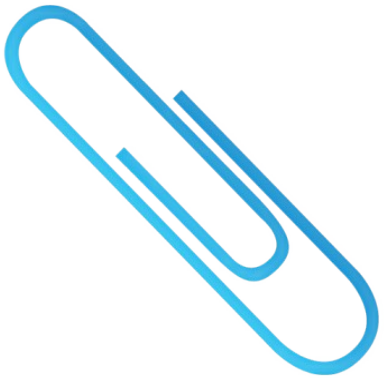
```
// Graphic.java
package graphics;
public abstract class Graphic {
    …
}

// Circle.java
package graphics;
public class Circle extends Graphic {
    …
}


//
// Three possibilities of package usage:
//

// Refer to package member.
graphics.Circle c = new graphics.Circle();

// Import package member.
import graphics.Rectangle;

// Import entire package.
import graphics.*;
```

```
java.applet.*
java.awt.*
java.beans.*
java.io.*
java.lang.*
java.math.*
java.net.*
java.nio.*
java.rmi.*
java.security.*
java.sql.*
java.text.*
java.util.*
javax.*
javax.accessibility.*
javax.crypto.*
javax.imageio.*
javax.net.*
javax.print.*
javax.rmi.*
javax.script.*
javax.security.*
javax.sound.midi.*
javax.sql.*
javax.swing.*
javax.tools.*
javax.transaction.*
javax.xml.*
org.w3c.dom.*
org.xml.sax.*
```

Code Organisation



Summary



- procedure: definition of algorithmic steps
- procedure call: invoke sub routine and return
- parameter forwarding: by value, by reference indirectly
- recursion: method calling itself
- method overloading: different parameters
- code organisation: package

Keywords

```
return

package
import
```

Contents

Motivation

Concept "Horse"

meta properties

happy, sad, aggressive

black, brown, white

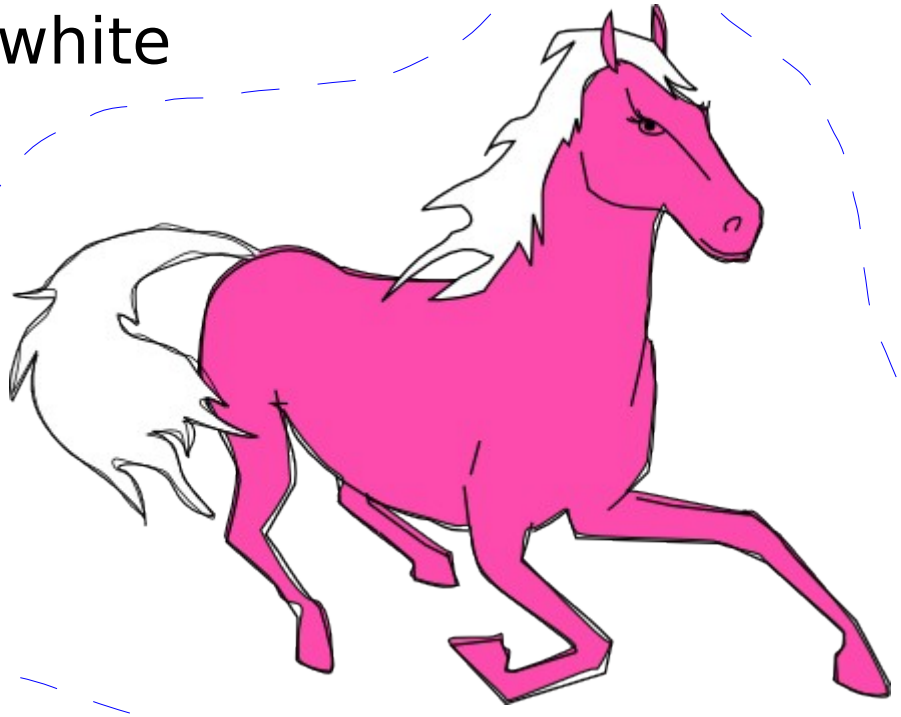
shape, size

smell

state structure

head, eyes, ears, mane

tail, legs, hoofs



external concepts

feed, hay, dung

ride, saddle, reins, stable

change logic

canter, gallop, tantivy

Class "Horse"

```
class Horse {

    void canter() {}
    void gallop() {}
    void tantivy() {}

    Head head;
    int legs;
    Tail tail;

    String mood;
    ColourSet colour;
    ShapeSet shape;
    Size size;
    SmellSet smell;
}

class Feed {

    Taste taste;
}

class Saddle {

    Material material;
}
```

change logic

state structure

meta properties

external concepts

Classification (Typification)

ClassName
attribute : Type
method(parameter : Type) : void

```
class ClassName {

    void method(Type: p) {}

    Type attribute;
}
```

Kinds of classes:
Concrete Class, Abstract Class, Interface

Instantiation (Object / Instance Creation)

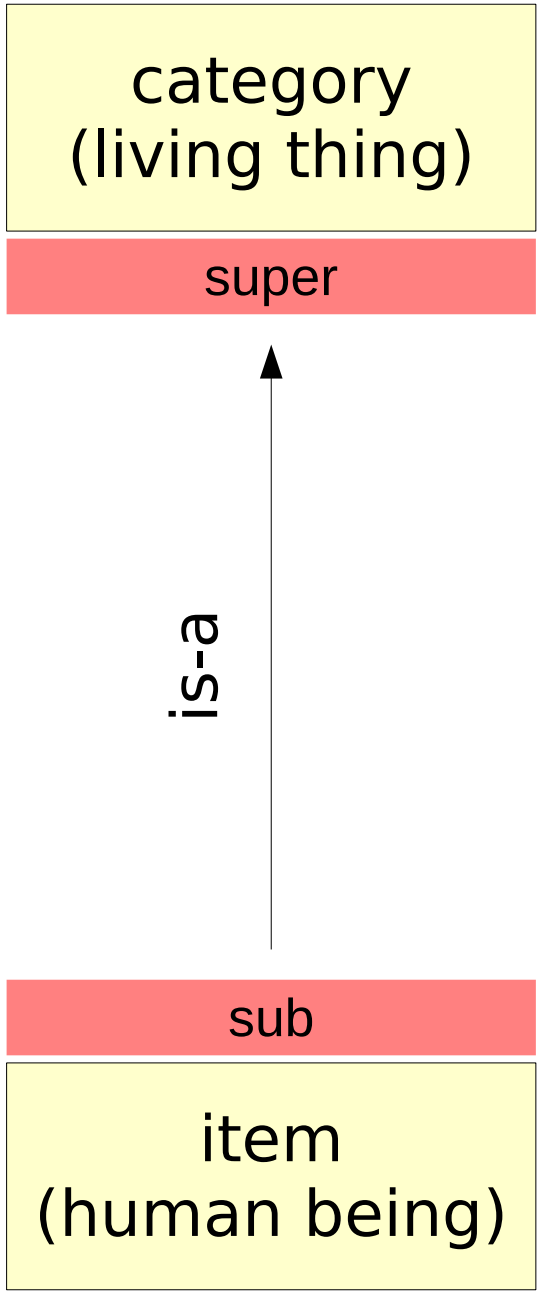
Memory
Stack Segment: - local / dynamic Variable / Parametre - return address at function calls
Heap/Data Segment: - object / instance - instance variable / object property - runtime value Each created object has its very own variable values (properties). Often, these are encapsulated and only manipulatable via access methods.
Code/Text Segment: - class / type structure / attribute type - method / instruction - static / global variable / constant - association / inheritance Static variables exist just once within an application and may be accessed globally (depending on visibility).

```
class Test {

    // A class attribute.
    static int s;

    // An instance attribute.
    int i;

    static void someMethod() {

        // Set static / global variable.
        Test.s = 10;

        // The objects / instance variables.
        Test t1 = new Test();
        Test t2 = new Test();

        // Set objects' property.
        t1.i = 20;
        t2.i = 30;

        // Print objects' property.
        System.out.println("t1: " + t1.i);
        System.out.println("t2: " + t2.i);
    }
}
```

Construction and Destruction

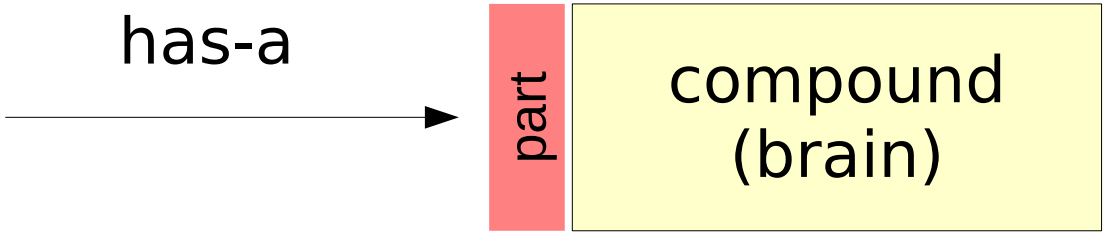
```
class Test {

    int i;

    Test() {

        // Set attribute.
        this.i = 0;
    }

    Test(int i) {

        // Set attribute.
        this.i = i;
    }
}
```

```
class Test2 {

    void someMethod() {

        // Create object.
        Test t1 = new Test();
        Test t2 = new Test(5);

        // Do something with the objects.
        System.out.println("t1.i: " + t1.i);
        System.out.println("t2.i: " + t2.i);

        // Mark objects as destroyable.
        t1 = null;
        t2 = null;
    }
}
```

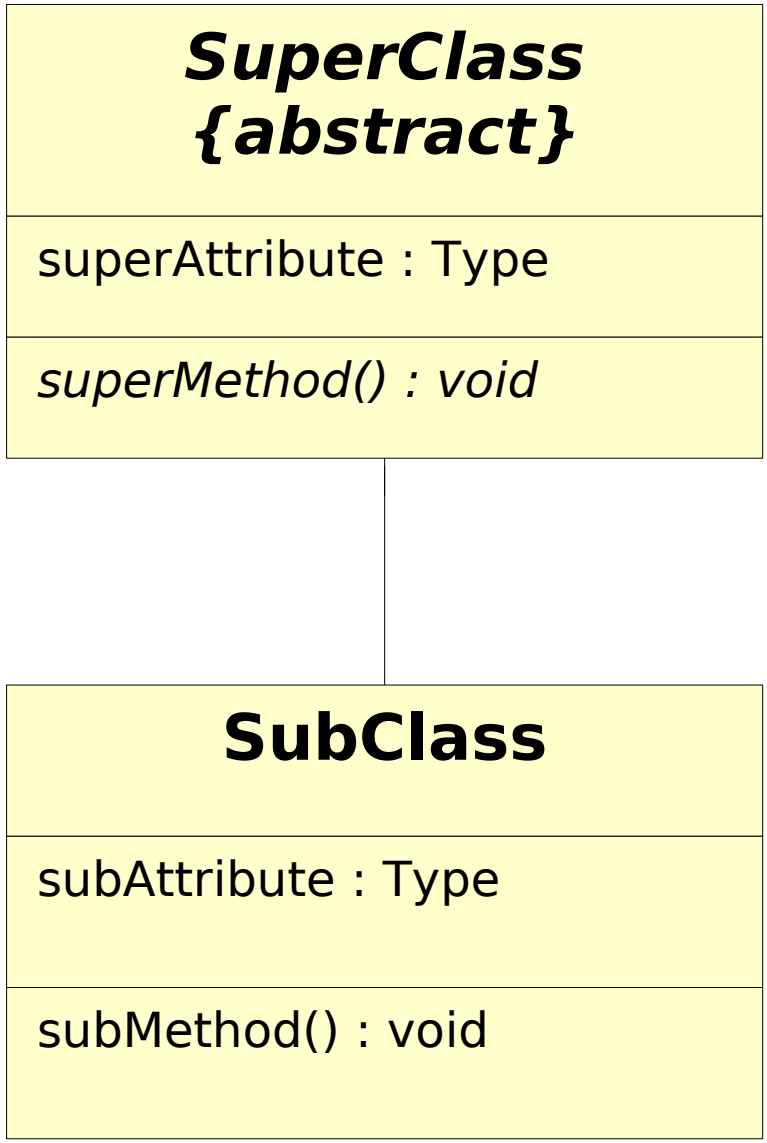


- Discrimination: on-/ offline thinking, 1st/ 2nd order, apes/ humans, self
- Categorisation: Socrates-Plato-Aristotle-Alexander; systematics of nature
- Composition: Leibnitz-Monades, micro-/ macrocosm, unidirectional

CAUTION! "parent-child" ambiguous



Inheritance




```
abstract class SuperClass {

    abstract void superMethod();

    Type superAttribute;
}
```

Vererbung Methoden von SuperClass

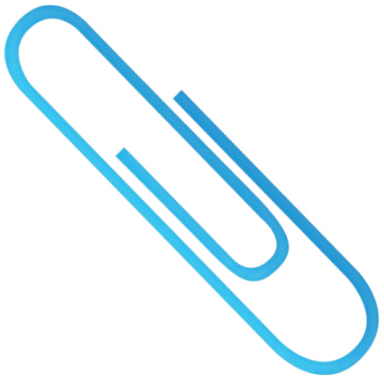
```
class SubClass extends SuperClass {

    // annotation helps detect spell errors
    // at compile time
    @Overwrite
    void superMethod() {

        // implementation
    }

    void subMethod() {

        // implementation
    }

    Type subAttribute;
}
```

Concrete Class / Abstract Class / Interface

IName   ***<<interface>>***
CONSTANT : Type
method(parameter : Type) : void

```
interface IName {

    static final Type CONSTANT = 5;

    void method(Type p);
}

class NameImpl implements IName {

    @Overwrite
    void method(Type p) {

        // implementation
    }
}
```

Summary



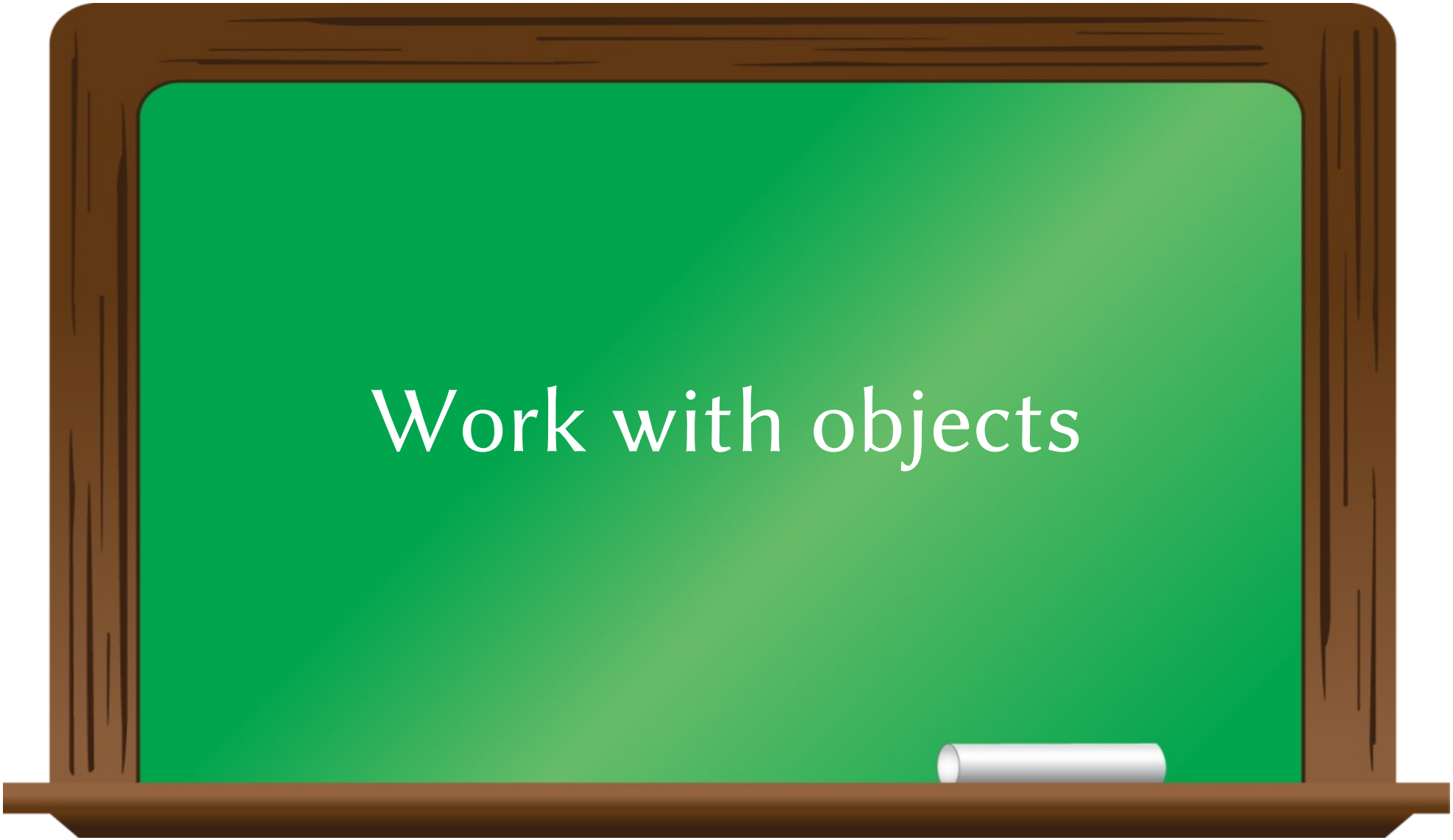
- OOP: abstract real world objects
- class: compound type owning attributes and methods
- object: instance of a class, constructed and destroyed
- inheritance: attributes and methods

Keywords

```
class
abstract
interface

extends
implements
final

static
null
```

Contents

Motivation



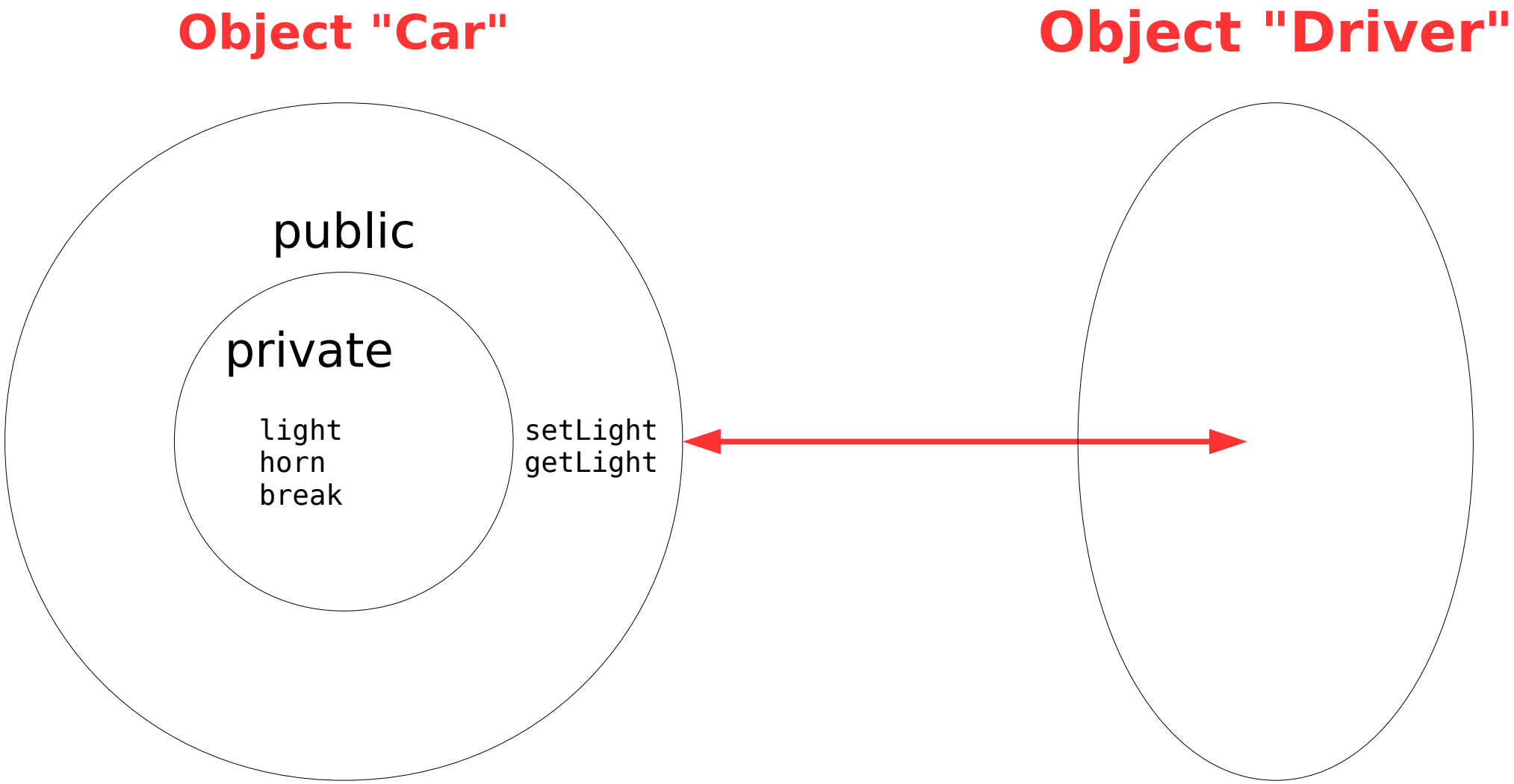
Encapsulation

ClassName
- attribute : Type - flag : boolean
+ setAttribute(param : Type) : void + getAttribute() : Type + setFlag(param : Type) : void + isFlag() : Type

```
public class ClassName {

    private Type attribute;
    private boolean flag;

    public void setAttribute(Type param) {

        this.attribute = param;
    }

    public Type getAttribute() {

        return this.attribute;
    }

    public void setFlag(boolean flag) {

        this.flag = flag;
    }

    public boolean isFlag() {

        return this.flag;
    }
}
```

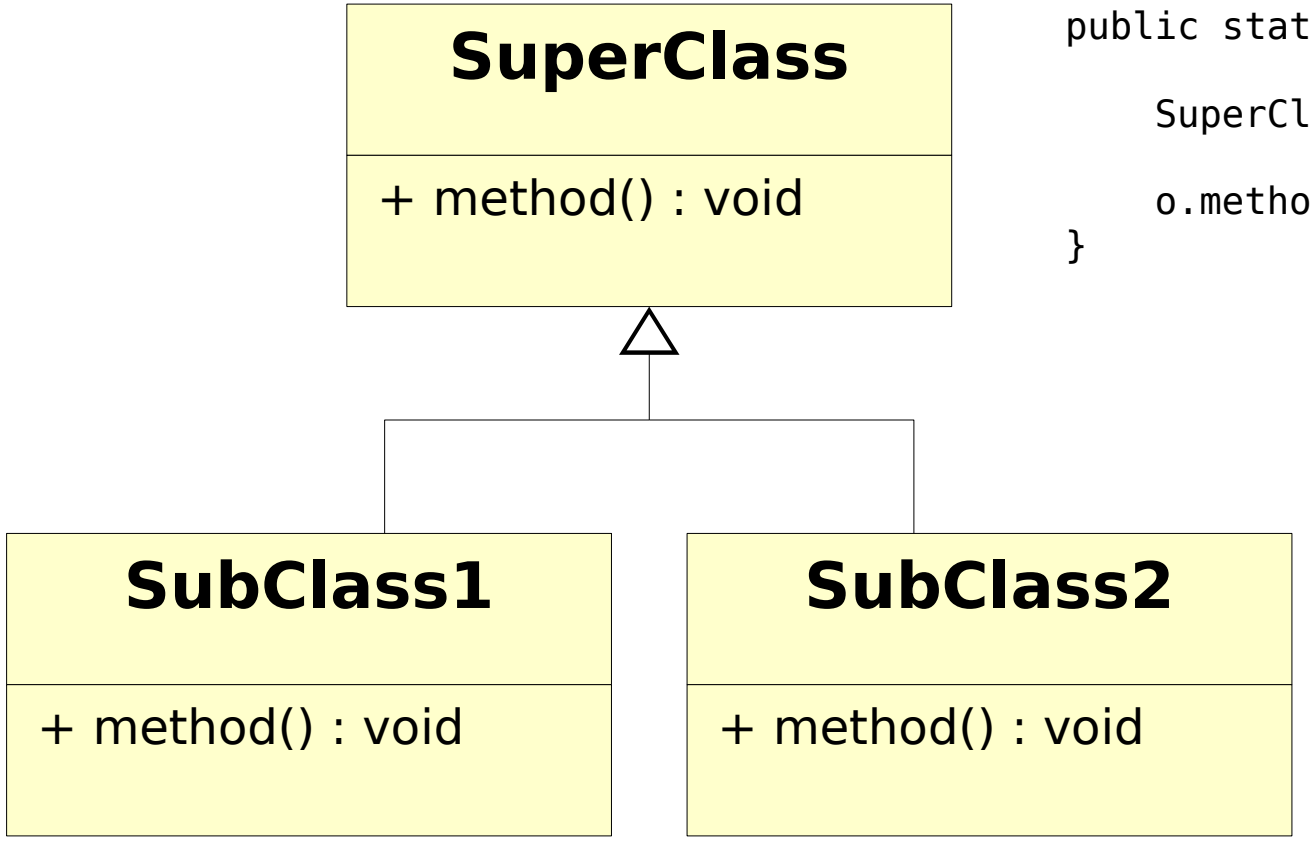
Access modifiers:
public, none (package), protected, private

Access Modifier



Public methods encapsulate private attributes

Polymorphism




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

    SuperClass o = new SubClass1();

    o.method();
}
```

Correct method (behaviour) is called, depending on instance

Polymorphism Exercise "Vehicle"

```
abstract class Vehicle {
    abstract int go();
}

class Bicycle extends Vehicle {
    int go() {
        return 5;
    }
}
class Car extends Vehicle {
    int go() {
        return 20;
    }
}

public class Launcher {
    public static void main(String[] args) {
        int d = 0; // The distance.
        Vehicle v = null; // The vehicle.
        v = new Bicycle();
        d = v.go();
        System.out.println("The first vehicle went a distance of: " + d + "\n");
        v = new Car();
        d = v.go();
        System.out.println("The second vehicle went a distance of: " + d + "\n");
    }
}
```

Summary

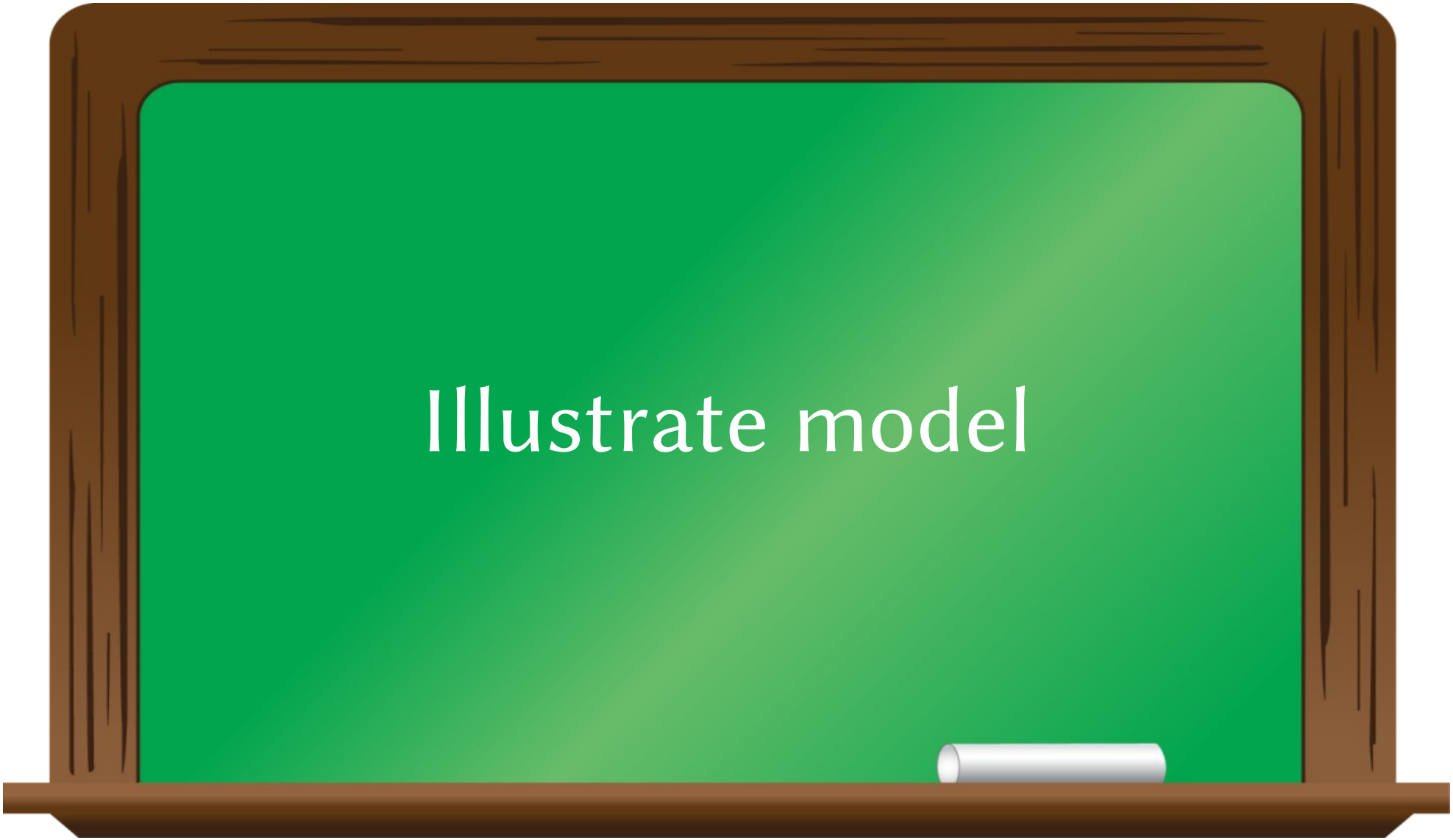
- encapsulation: protect attributes with access modifier
- access: via public methods to manipulate values
- polymorphism: work with super type of a variable
- sub class method: called by interpreter
- static: overloading
- dynamic: overriding

Keywords

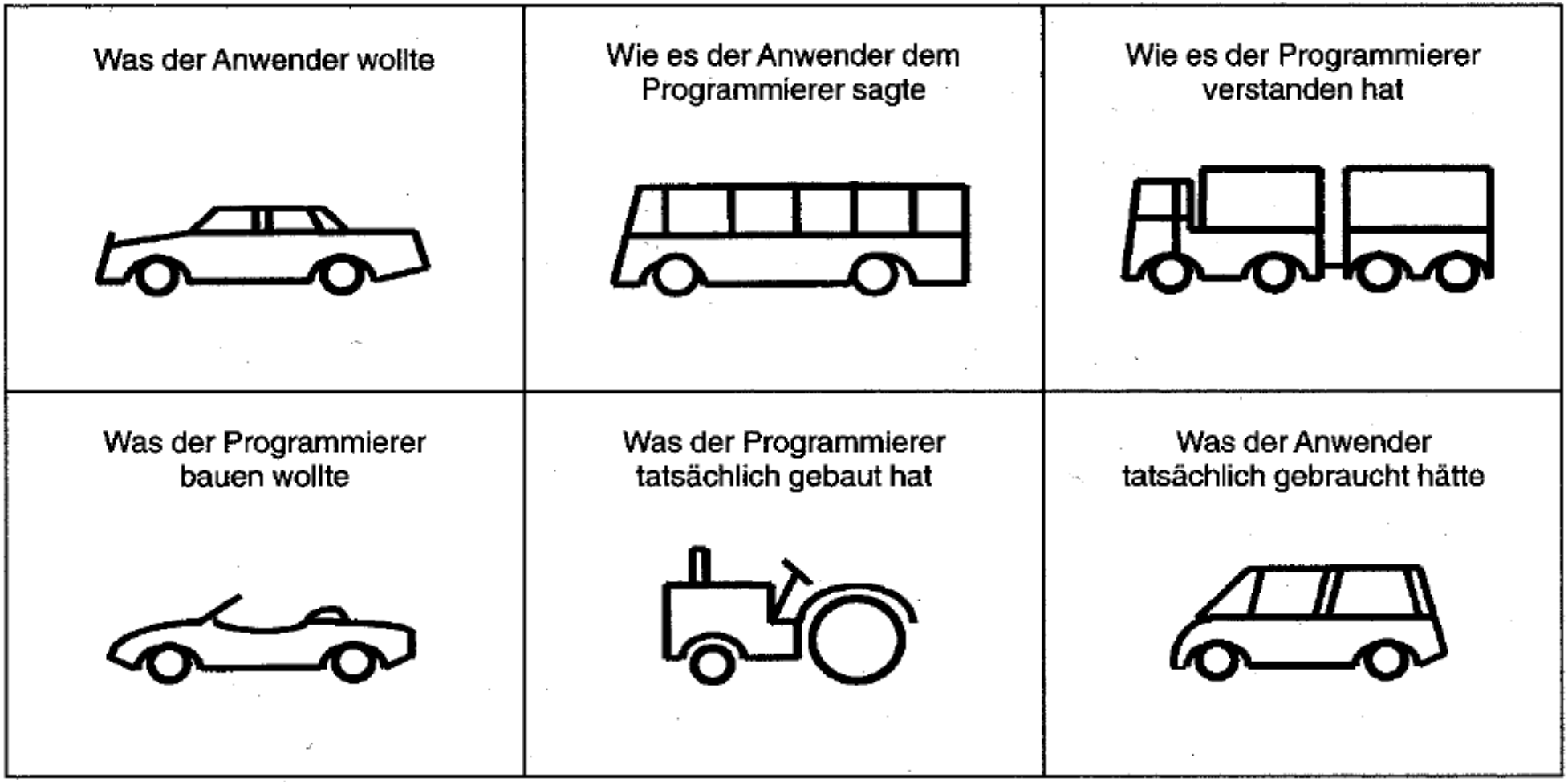
```
public
protected
private
```

Contents

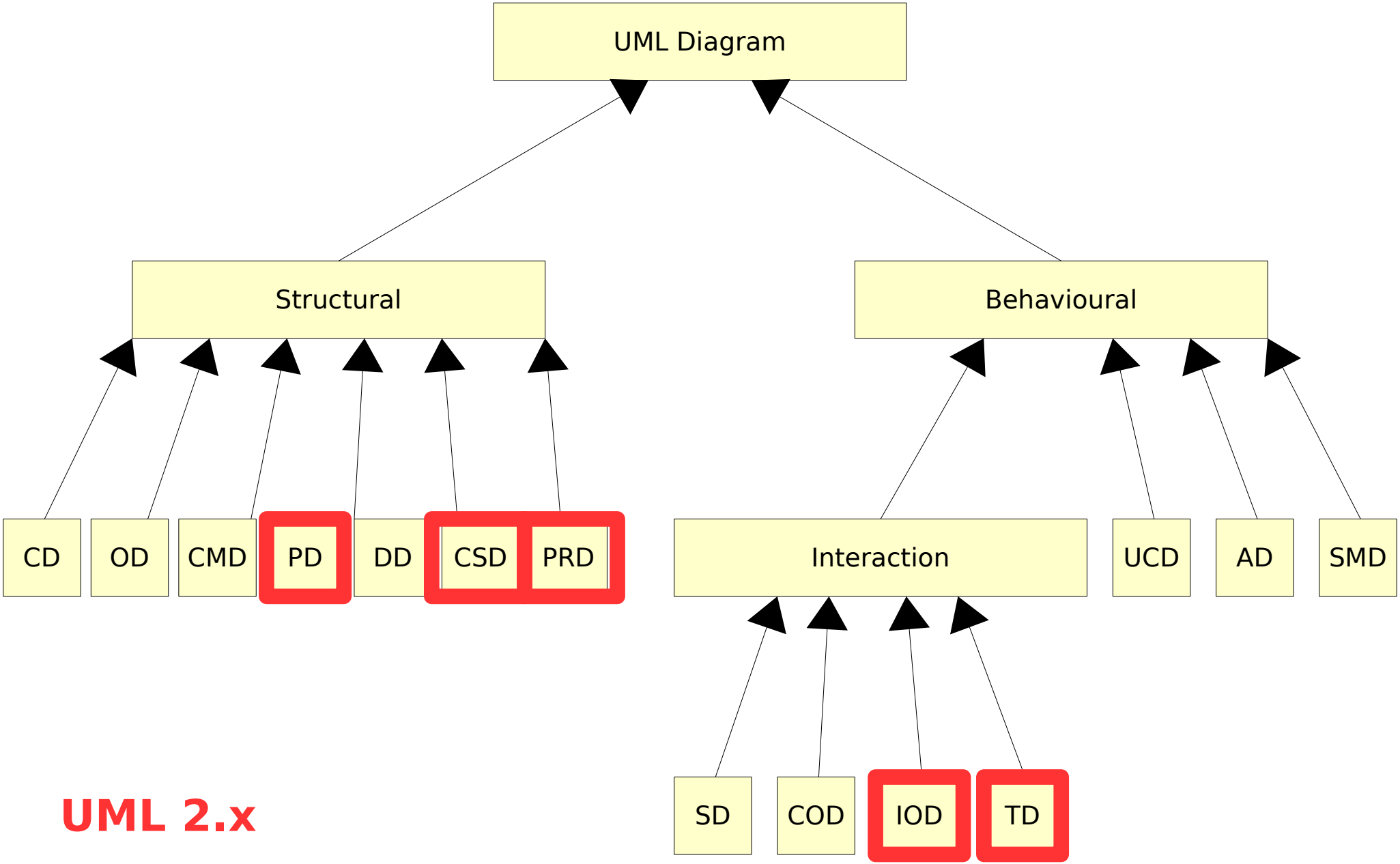
Motivation



Motivation

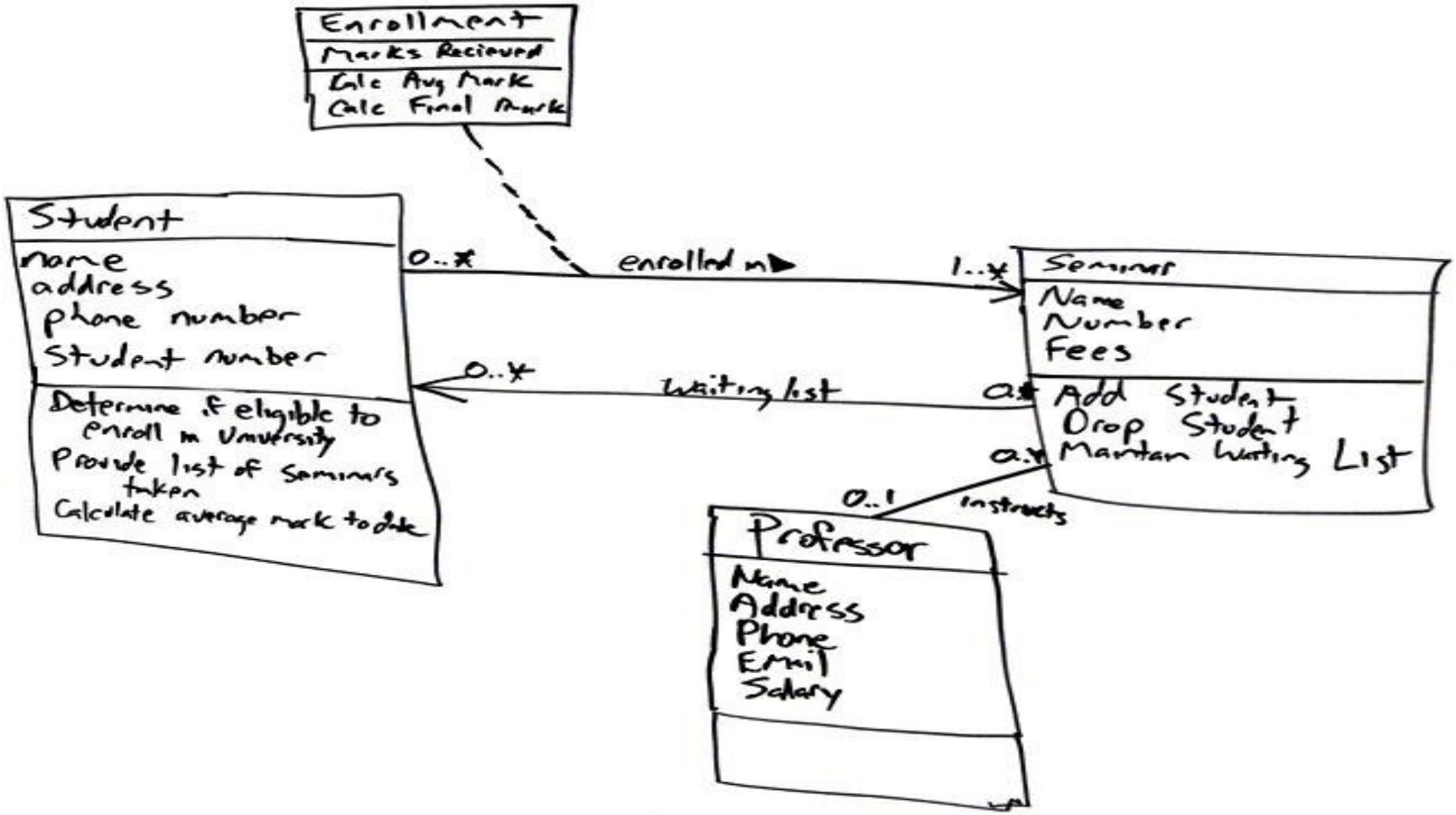


Johannes Siedersleben. Softwaretechnik: Praxiswissen für Softwareingenieure. Hanser, 2002


UML Diagram
Structural
Behavioural
CD
OD
CMD
PD
DD
CSD
PRD
Interaction
UCD
AD
SMD
SD
COD
IOD
TD

UML 2.x

Manual Modelling

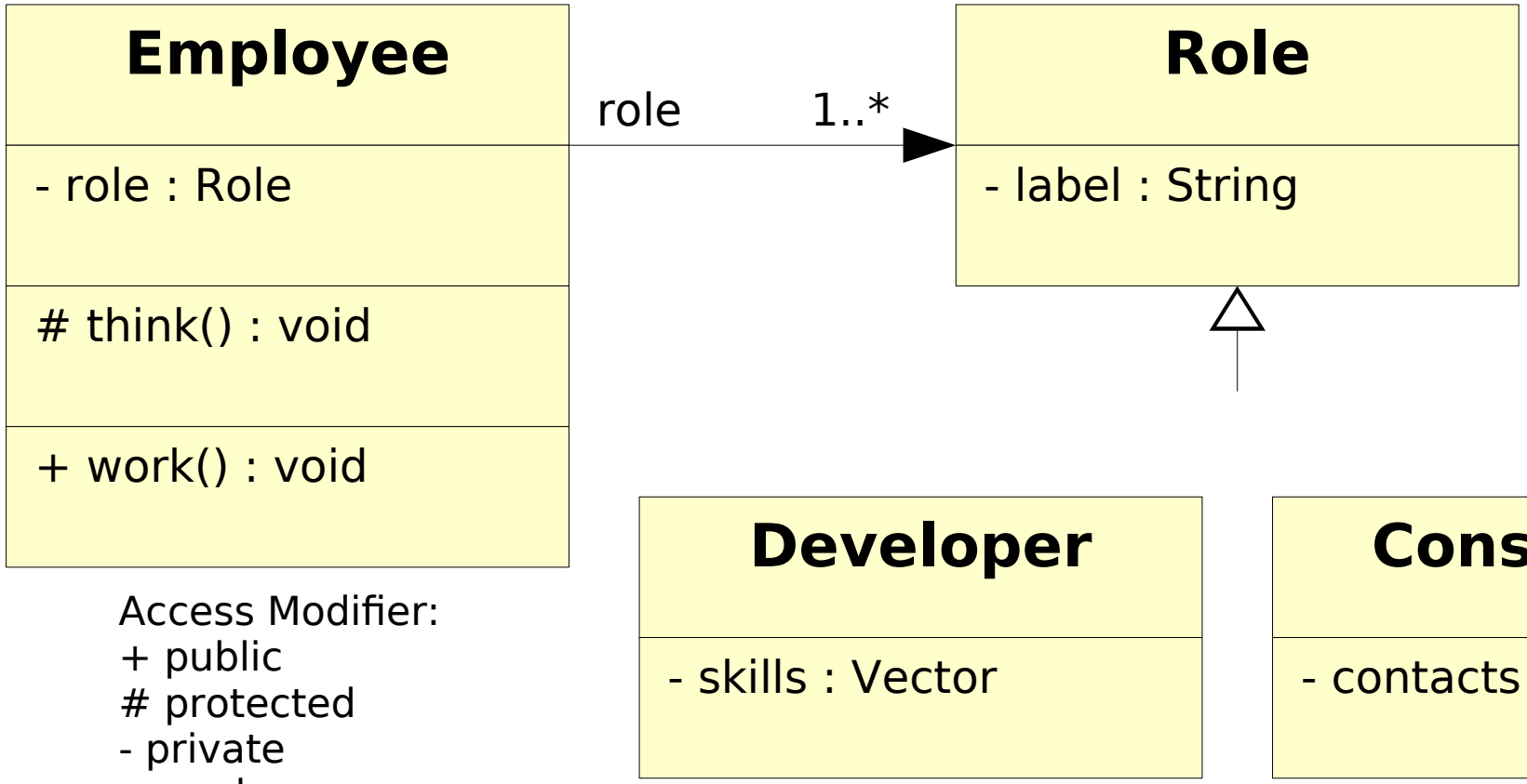


http://www.agilemodeling.com/artifacts/classDiagram.htm

Class Diagram (CD)

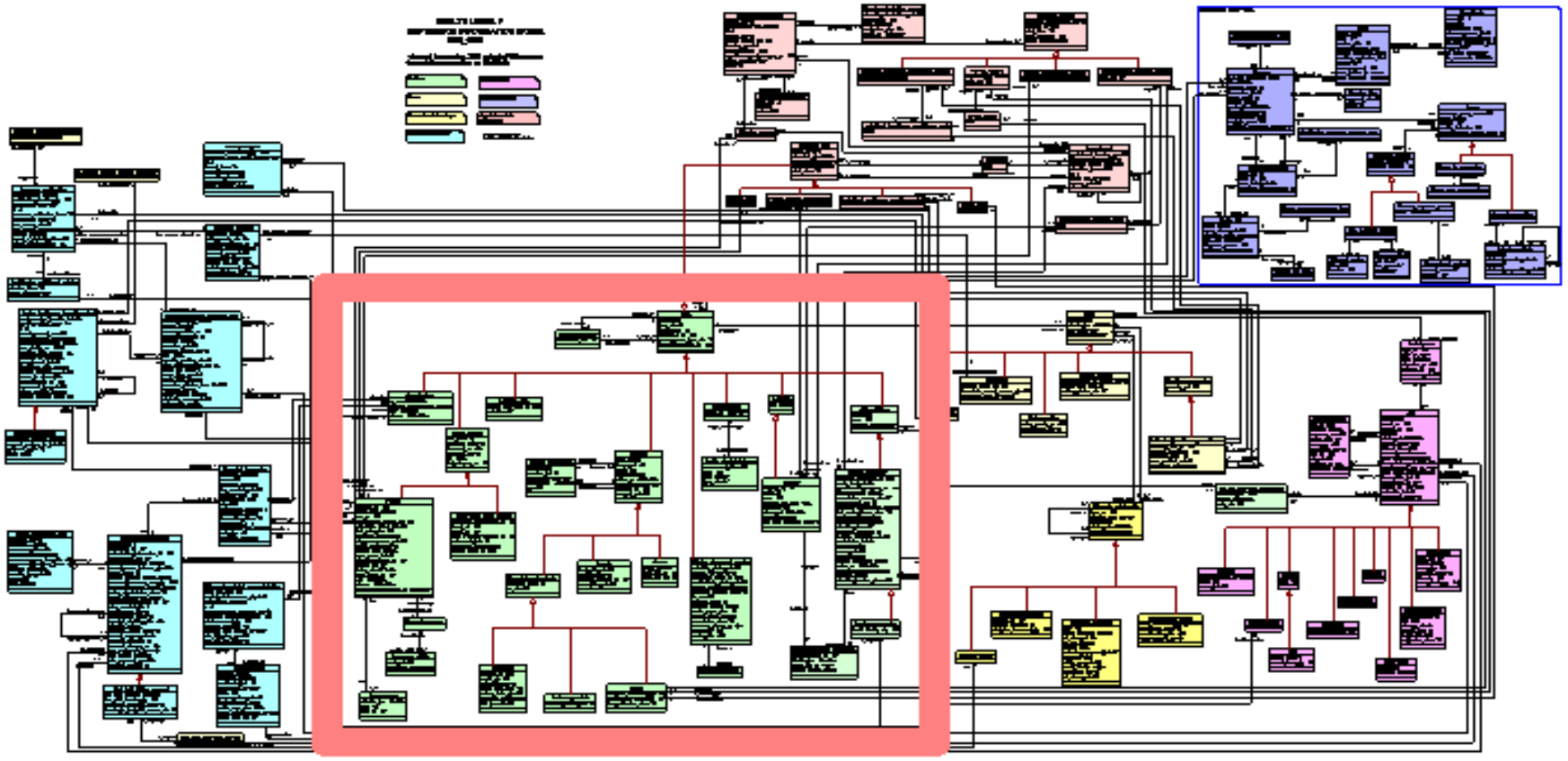
Class:
- name
- attributes
- methods

Association:
- name
- direction
- cardinality (multiplicity)



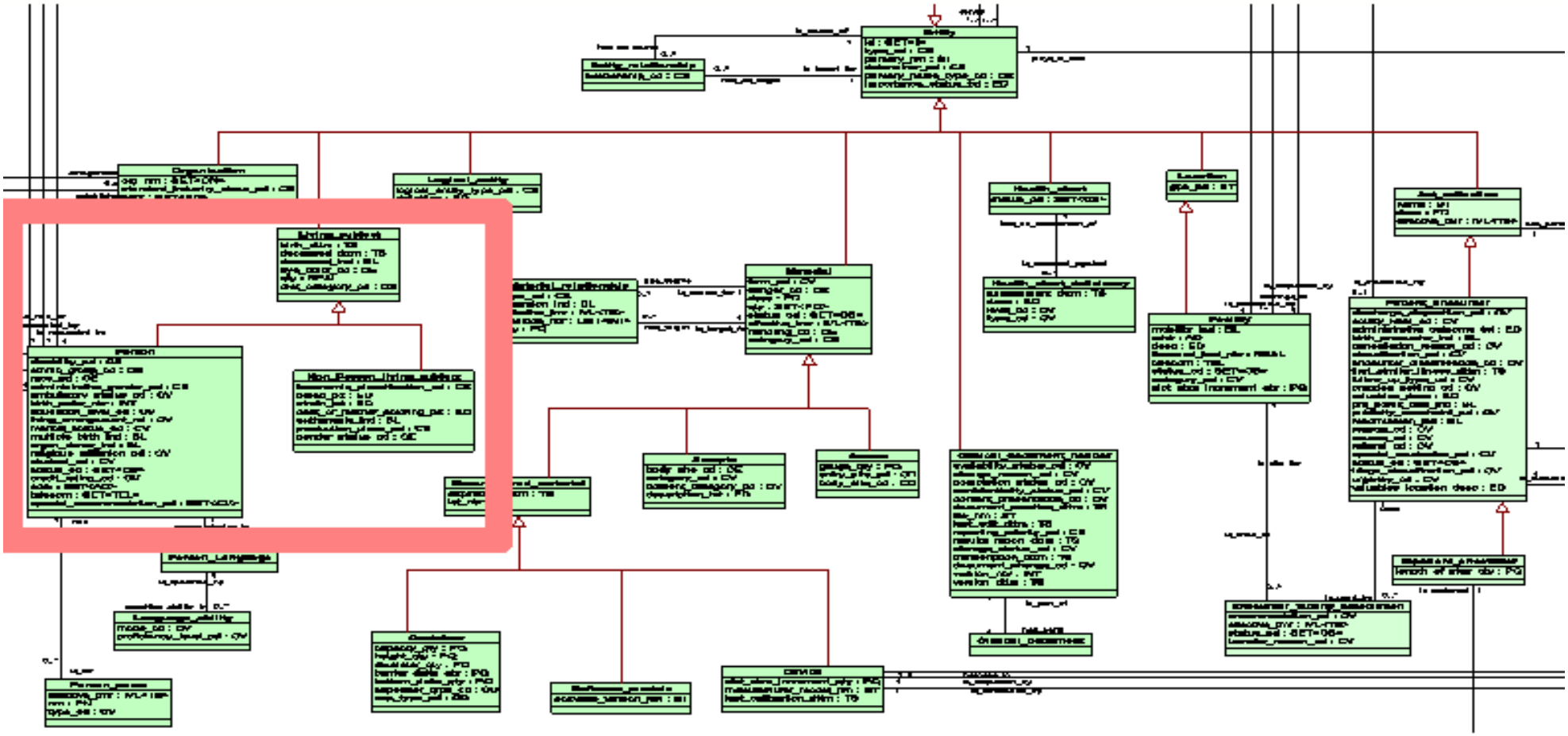
Inheritance:
- super class
- sub classes

Access Modifier:
+ public
# protected
- private
~ package



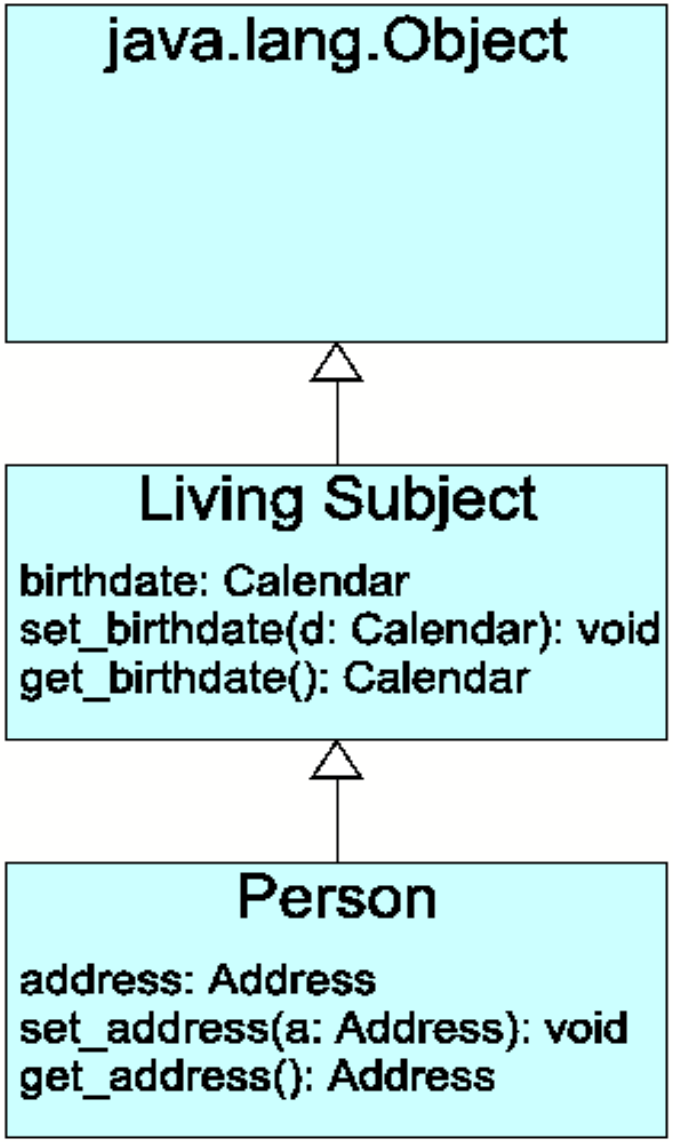
Health Level 7 - RIM Reference Information Model	scheduling	message
finance	role	service
entity	role-role-relation	

http://www.hl7.org/implement/standards/rim.cfm



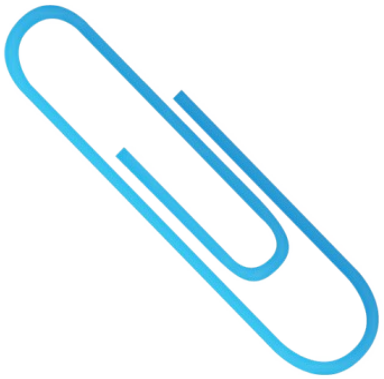
- Entity: Organisation, Living Subject, Material, Health Chart, Location
- Living Subject: Person, Non Person

http://www.hl7.org/implement/standards/rim.cfm




```
Address address;

void set_address(Address a) {

    this.address = a;
}

Address get_address() {

    return this.address;
}
```

Summary



- language: graphical notation
- diagram types: structural, behavioural
- class diagram: structure and relations between classes
- software pattern: reusable constellation of classes

Contents

Motivation

Avoid programme crash

Exception

```
christian@deneb:~$ java NonExistingClass
Exception in thread "main" java.lang.NoClassDefFoundError: NonExistingClass
Caused by: java.lang.ClassNotFoundException: NonExistingClass
        at java.net.URLClassLoader$1.run(URLClassLoader.java:217)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:205)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:319)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:264)
Could not find the main class: NonExistingClass. Program will exit.
```

= unexpected (abnormal) condition in an application
= event occurring during the execution of a program, that disrupts the normal flow of instructions

Handling

```
File f = Launcher.openFile();

try {

    Launcher.readFile(f);

} catch (NewException e1) {

    System.out.println("Caught exception: " + e1.getMessage());

} catch (NewException2 e2) {

    System.out.println("Caught exception: " + e2.getMessage());

} catch (NewException3 e3) {

    System.out.println("Caught exception: " + e3.getMessage());

} catch (Exception e) {

    System.out.println("Caught general exception: " + e.getMessage());

} finally {

    Launcher.closeFile(f);
}
```

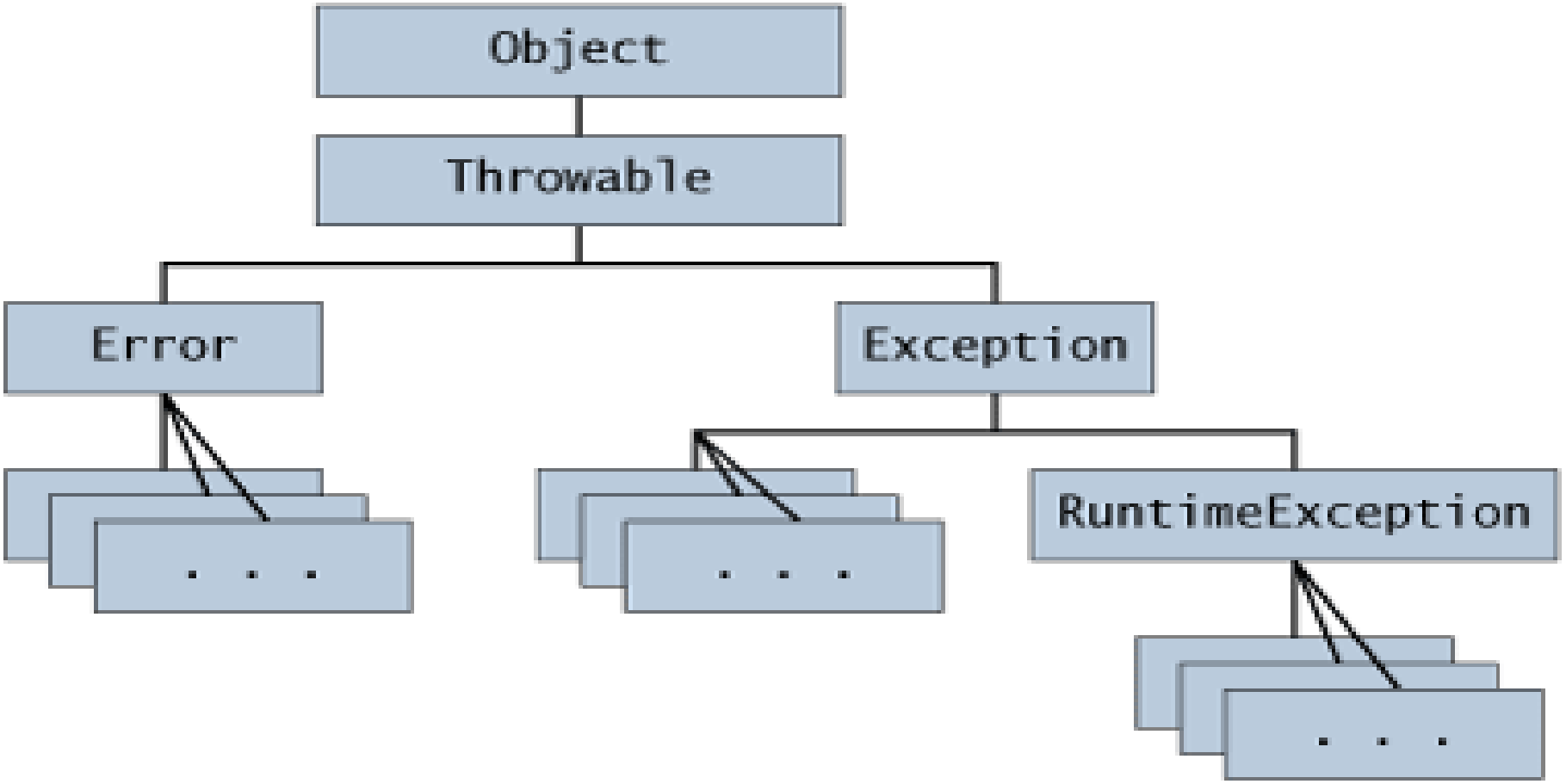
Error Management Techniques

```
void read_stream_socket_data(void* p0, void* p1) {

    void** b = (void**) p0;
    // Initialise error number. It is a global variable
    // and other operations may have set some value
    // that is not wanted here.
    errno = 0;
    // Receive data. Set buffer count.
    int bc = recv(ps, b, bs, 0);

    if (bc > 0) {
        log_message(L"Success!");
    } else if (bc == 0) {
        log_message(L"No data.");
    } else {
        if (errno == EBADF) {
            log_message(L"Could not ...");
        } else if (errno == ENOTSOCK) {
            log_message(L"Could not ...");
        } else if (errno == EWOULDBLOCK) {
            log_message(L"Could not ...");
        } else {
            log_message(L"Could not ...");
        }
    }
}
```

Kinds of Exceptions

Checked by compiler	Unchecked by compiler	Error
java.lang. Exception	**java.lang. RuntimeException**	**java.lang. Error**

Exception Object



https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html

Exception Throwing

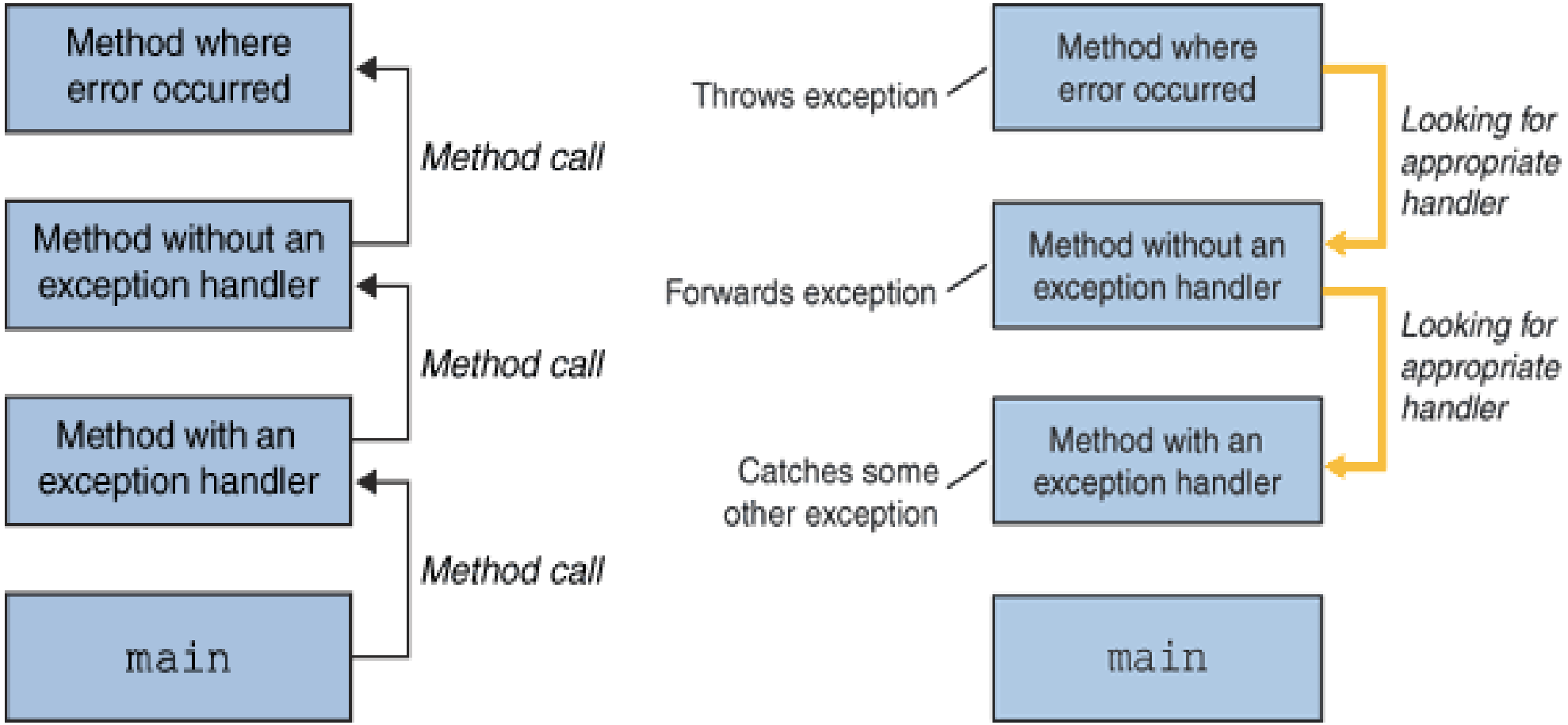
```
class Test {

    static double divide(int a, int b) throws Exception {

        double r = 0.0;

        if (b != 0) {
            r = a / b;
        } else {
            throw new Exception("Error: Division by zero!");
        }

        return r;
    }

    static void main(String[] a) {

        try {
            double r = Test.divide(4, 0);
            System.out.println("The result is: " + r);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Exception Forwarding

```
class Test {

    static double divide(int a, int b) throws MyException {
        if (b != 0) {
            return a / b;
        } else {
            throw new MyException("Error: Division by zero!");
        }
    }

    static double calculate() throws MyException {
        return Test.divide(4, 0);
    }

    static void main(String[] a) {
        try {
            double r = Test.calculate();
            System.out.println("The result is: " + r);
        } catch (MyException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

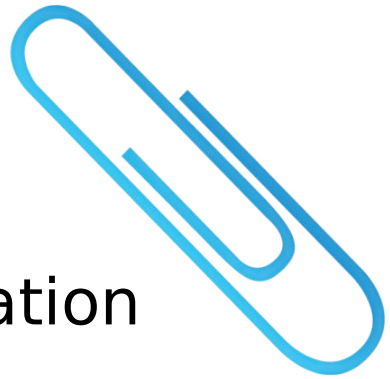
```
class MyException extends Exception {

    public MyException(String s) {
        super(s);
    }
}
```

Searching Call Stack for Exception Handler



https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

Summary



- exception: unexpected condition in an application
- kinds: checked, unchecked (error, runtime)
- hierarchy: Object-Throwable-Exception-RuntimeException

Keywords

```
try
catch
finally
throw
throws
```