12.–15.09.2010
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Do you Unicode?

Details zu Unicode und der Anwendung in moderner Software

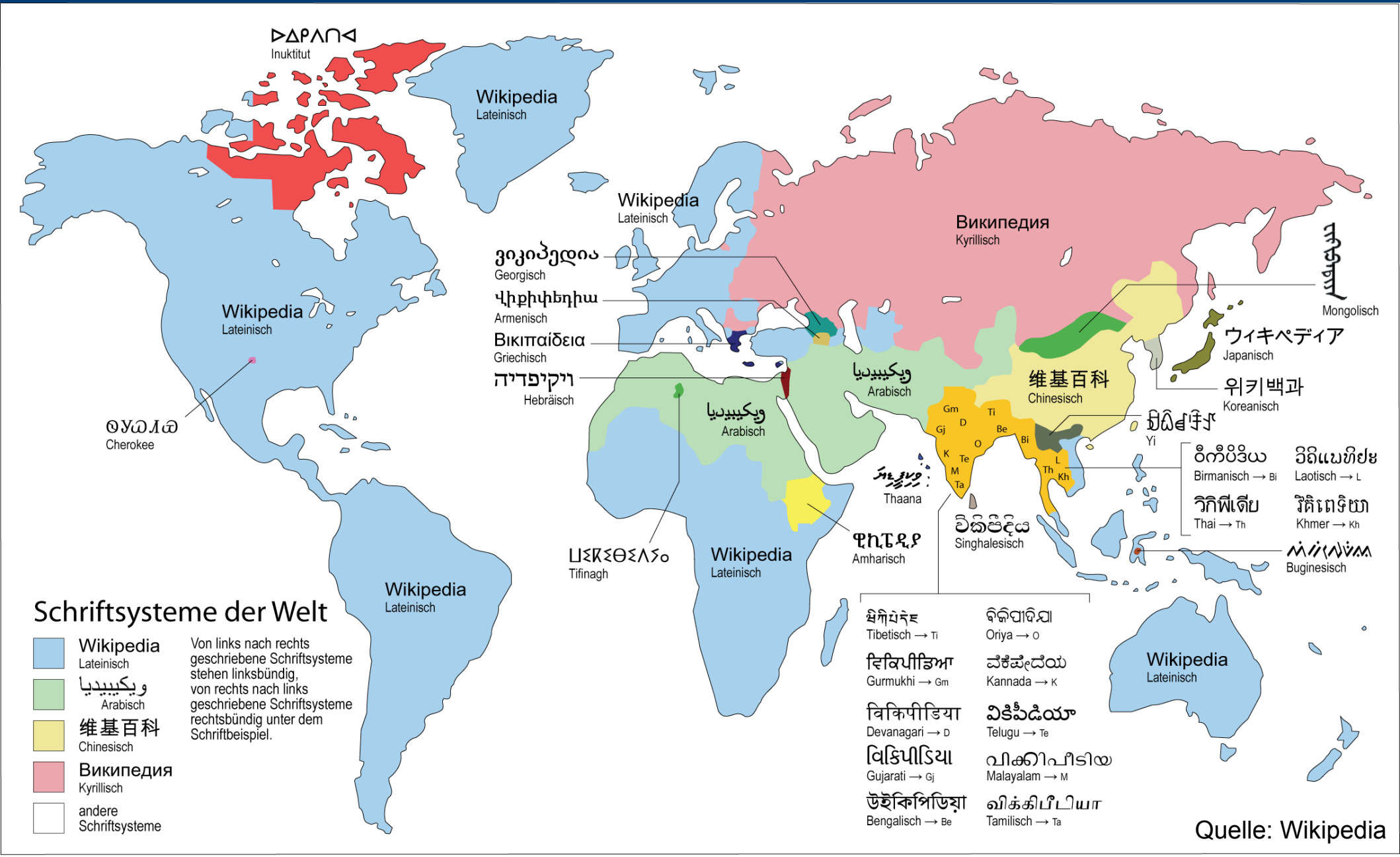
Karol Rückschloss

MATHEMA Software GmbH

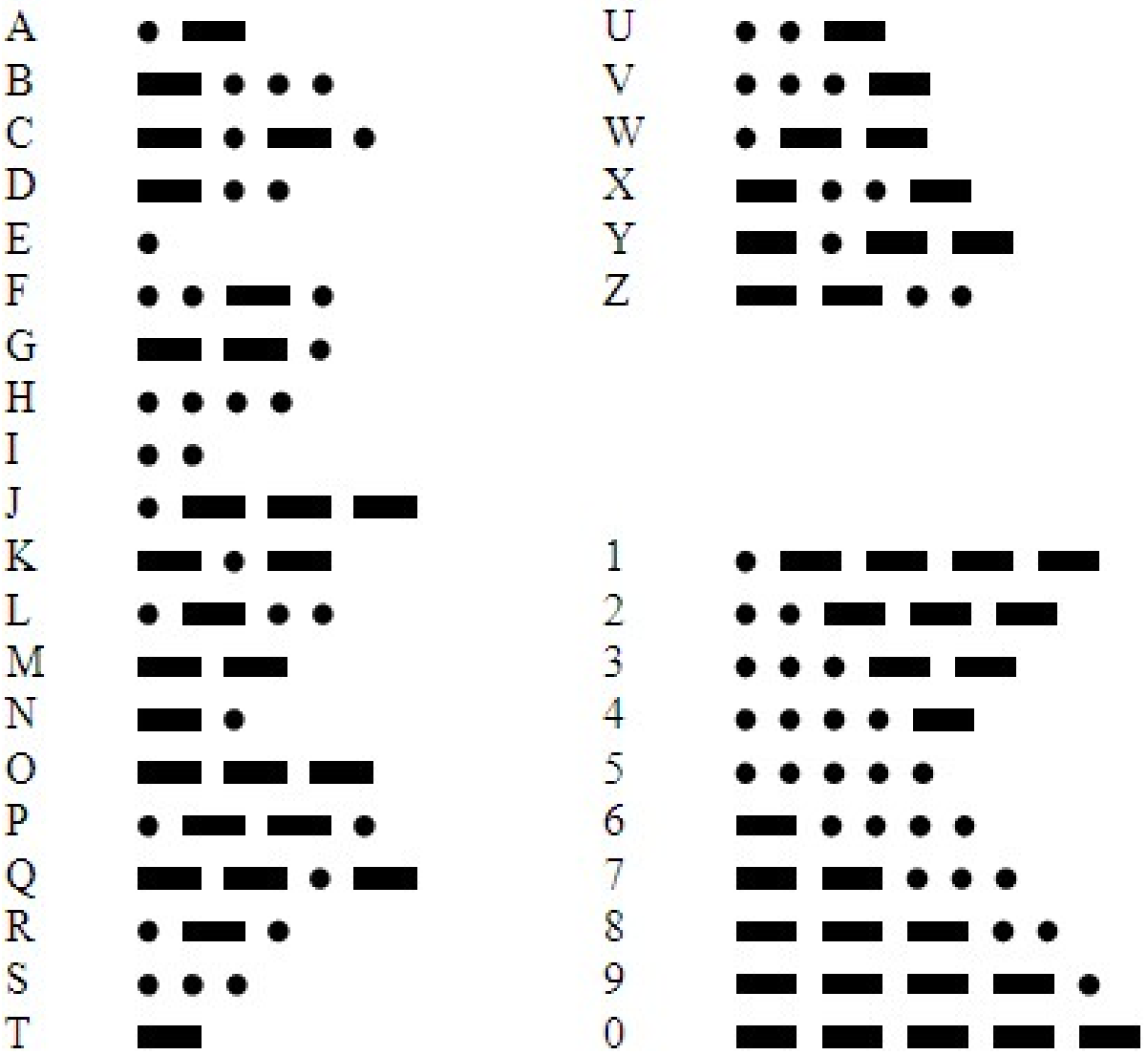
Agenda

- **Einführung**
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- **Struktur**
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- **Eigenschaften und Operationen**
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- **Encodings**
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- **Unicode in Java**
- **Weitere Informationsquellen**

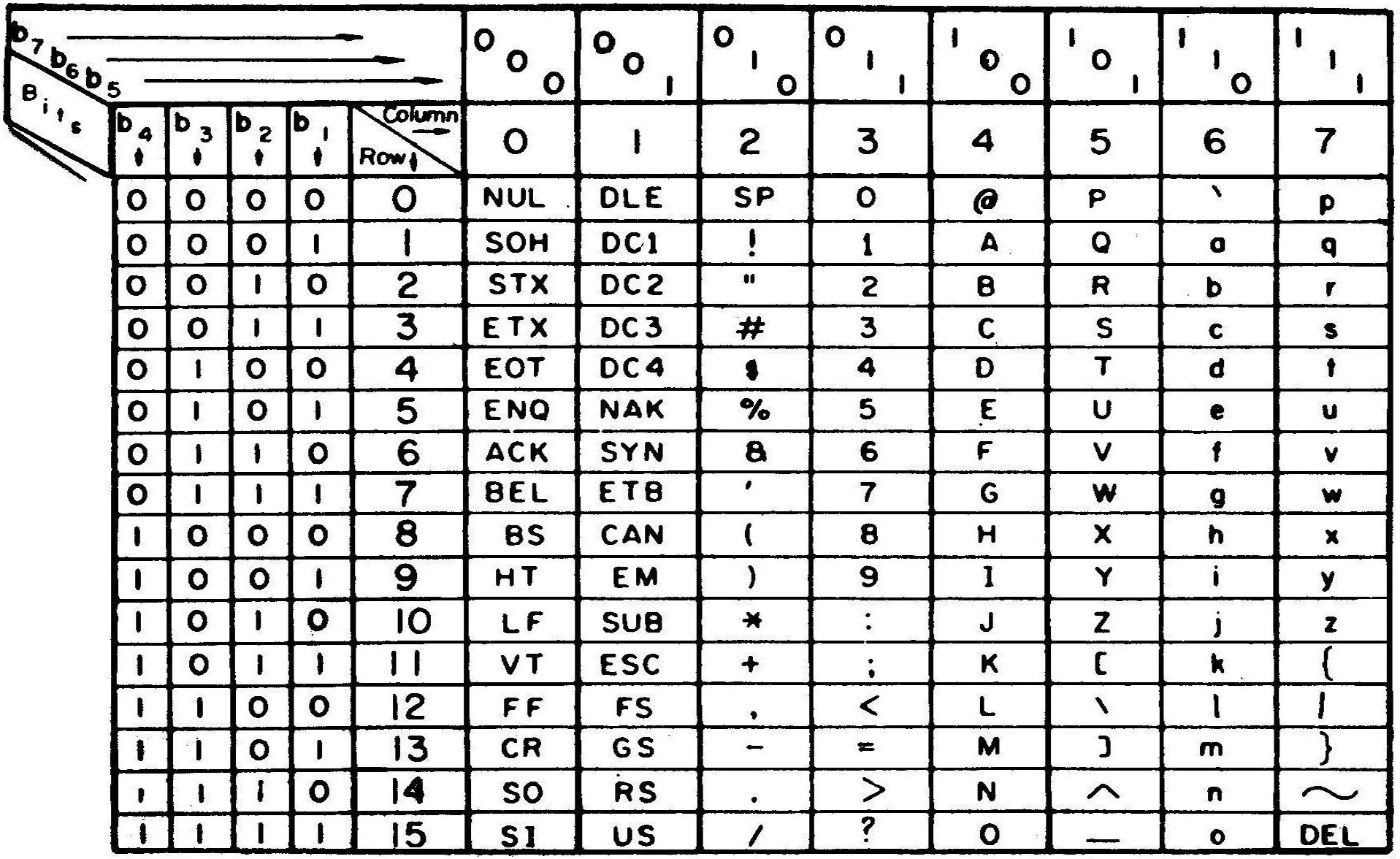
- **Einführung**
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- Struktur
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- Eigenschaften und Operationen
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- Encodings
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- Unicode in Java
- Weitere Informationsquellen



Das Morse-Alphabet



ASCII (1968)



b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

- 7-Bit ASCII nur Basiszeichen
- Latin 1: ASCII + Deutsch, Spanisch, Französisch, ...

- Andere Zeichen mit Diakritik?
- Latin 2: ASCII + mittel- und osteuropäische Sprachen

8-Bit Codes

- Für jede Sprache oder Sprachgruppe eigener 8-Bit Code

- Byte-Wert E0 bedeutet:

  à in ISO 8859-1 (Westeuropa)

  ŕ in ISO 8859-2 (Zentral-/Osteuropa)

  א in ISO 8859-8 (Hebräisch)

Mädchen möchten Müsli

➔ Мдdchen mцchten Mьsli

➔ Мфdchen mЎchten M№sli

UNICODE

Þó er möguleiki en þó aðeins í samráði við ritstjóra

ليسا ةيفحصلا نييقارعلا نييفحصلا ةباقن

今天上午，国家统计局发布的最新宏观经济数据显示

Предпосылки создания и развитие Юникода

∀x ∈ ℝ: ⌈x⌉ = − ⌊−x⌋ , α ∧ ¬β = ¬(¬α ∨ β)

♠♣♥♦ ❦ ☲☲☲☲ ♫♬ ☺☹☠☮

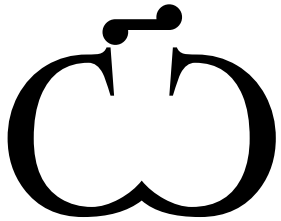
flēißigę ♪

- Einführung
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- **Struktur**
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- Eigenschaften und Operationen
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- Encodings
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- Unicode in Java
- Weitere Informationsquellen

ä

Name: LATIN SMALL LETTER A WITH DIAERESIS

Codepoint: U+00E4



Name: CYRILLIC CAPITAL OMEGA WITH TITLO

Codepoint: U+047C

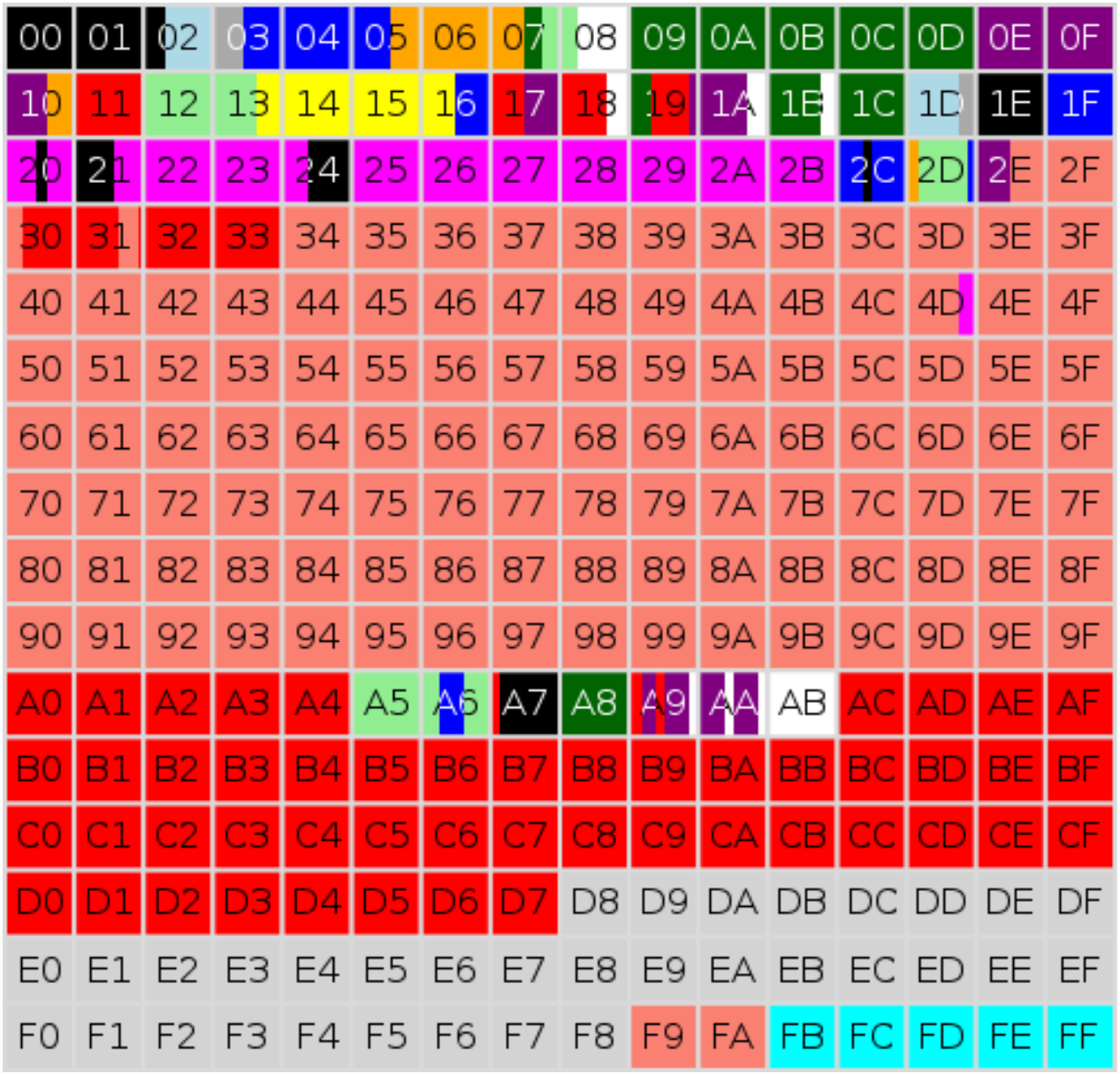
Unicode Codespace

- Codespace von 1.114.112 Codepoints (0 bis $10FFFF_{16}$)
- U+ Notation: U+009A, U+2211, U+102CD4
- Kategorien von Codepoints:
  - Grafische Zeichen (a, Đ, ų, Σ, ش, 裴 , ♣, ...)
  - Surrogates (für UTF-16)
  - Noncharacters
  - Reservierte Zeichen (spätere Unicode-Versionen)
  - Private Zeichen (keine Unicode Interpretation)
  - Formatzeichen (beeinflussen Anzeige anderer Zeichen)
  - Kotrollzeichen (Tab, CR, LF, Schreibrichtung, ...)

Codepoints sind in 17 Ebenen (Planes) unterteilt

Plane	Bereich	Beschreibung	Abkürzung
0	0000–FFFF	**Basic Multilingual Plane**	**BMP**
1	10000–1FFFF	Supplementary Multilingual Plane	SMP
2	20000–2FFFF	Supplementary Ideographic Plane	SIP
3	30000–3FFFF	Tertiary Ideographic Plane	TIP
4 bis 13	40000–DFFFF	momentan nicht zugewiesen	
14	E0000–EFFFF	Supplementary Special-purpose Plane	SSP
15	F0000–FFFFF	Supplementary Private Use Area-A	
16	100000–10FFFF	Supplementary Private Use Area-B	

Basic Multilingual Plane (BMP)



Blöcke

- Codepoints sind in 197 Blöcken organisiert
- Beispiele:

0000..007F; Basic Latin
0080..00FF; Latin-1 Supplement
0100..017F; Latin Extended-A
0180..024F; Latin Extended-B
0250..02AF; IPA Extensions
02B0..02FF; Spacing Modifier Letters
0300..036F; Combining Diacritical Marks
0370..03FF; Greek and Coptic
0400..04FF; Cyrillic
0500..052F; Cyrillic Supplement
0530..058F; Armenian
0590..05FF; Hebrew
0600..06FF; Arabic
0700..074F; Syriac

10280..1029F; Lycian
102A0..102DF; Carian
10300..1032F; Old Italic
10330..1034F; Gothic
10380..1039F; Ugaritic
103A0..103DF; Old Persian
10400..1044F; Deseret
10450..1047F; Shavian
10480..104AF; Osmanya
10800..1083F; Cypriot Syllabary
10840..1085F; Imperial Aramaic
10900..1091F; Phoenician
10920..1093F; Lydian
10A00..10A5F; Kharoshthi

- Für jeden Block gibt es eine entsprechende Code Chart

European Scripts
Armenian
Armenian Ligatures
Coptic
Coptic in Greek block
Cypriot Syllabary
Cyrillic
Cyrillic Supplement
Cyrillic Extended-A
Cyrillic Extended-B
Georgian
Georgian Supplement
Glagolitic
Gothic

African Scripts
Bamum
Egyptian Hieroglyphs (1MB)
Ethiopic
Ethiopic Supplement
Ethiopic Extended
N'Ko
Osmanya
Tifinagh
Vai
Middle Eastern Scripts
Arabic
Arabic Supplement
Arabic Presentation Forms-A

Code Chart für Block „Cyrillic“ (0400-04FF)

	040	041	042	043	044	045
0	Ѐ 0400	А 0410	Р 0420	а 0430	р 0440	ѐ 0450
1	Ё 0401	Б 0411	С 0421	б 0431	с 0441	ё 0451
2	Ђ 0402	В 0412	Т 0422	в 0432	т 0442	ђ 0452
3	Ѓ 0403	Г 0413	У 0423	г 0433	у 0443	ѓ 0453
4	Є 0404	Д 0414	Ф 0424	д 0434	ф 0444	є 0454

Cyrillic extensions

0400 Ѐ CYRILLIC CAPITAL LETTER IE WITH GRAVE
≡ 0415 Е 0300 ◌̀

0401 Ё CYRILLIC CAPITAL LETTER IO
≡ 0415 Е 0308 ◌̈

0402 Ђ CYRILLIC CAPITAL LETTER DJE

0403 Ѓ CYRILLIC CAPITAL LETTER GJE
≡ 0413 Г 0301 ◌́

0404 Є CYRILLIC CAPITAL LETTER UKRAINIAN IE

0405 Ѕ CYRILLIC CAPITAL LETTER DZE

0406 І CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I
→ 0049 I latin capital letter i
→ 0456 і cyrillic small letter byelorussian-ukrainian i
→ 04C0 Ӏ cyrillic letter palochka

„flēißigę ♪“ in Code Charts

FB02 ﬂ LATIN SMALL LIGATURE FL
≈ 0066 f 006C l

00DF ß LATIN SMALL LETTER SHARP S
= Eszett
- German
- uppercase is “SS”
- in origin a ligature of 017F ſ and 0073 s

→ 03B2 β greek small letter beta
→ 1E9E ẞ latin capital letter sharp s

- Einführung
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- Struktur
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- **Eigenschaften und Operationen**
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- Encodings
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- Unicode in Java
- Weitere Informationsquellen

Die 10 Designprinzipien von Unicode

1) **Allgemeingültigkeit (Universality):** ein einziges Repertoir von Zeichen für allgemeine Nutzung
2) **Effizienz:** Unicode-Text soll einfach zu verarbeiten sein
3) **Zeichen, nicht Glyphen:** Codepoints = Abstraktionen, nicht visuellen Erscheinung der Zeichen
4) **Semantik:** Zeichen haben eine klar definierte Bedeutung, gemeint sich hauptsächlich Eigenschaften wie Spacing, Kombinierbarkeit oder Schreibrichtung, nicht die Bedeutung an sich (was bedeutet Ω ?)
5) **Einfacher Text:** Unicode behandelt einfachen Text und nicht Formatierung, bis auf Sachen wie Zeichenumbrüche

Die 10 Designprinzipien von Unicode

6) **Logische Anordnung:** z.B. dass diakritische Zeichen nach dem Basiszeichen kommen, auch wenn visuell anders: Ὼ

7) **Vereinheitlichung (Unification):** Zeichenduplikate als ein Codepoint – dänisches und isländisches æ beides U+00E6

8) **Dynamische Komposition:** Zeichen können dynamisch zusammengebaut werden, z.B. ä = a + ¨ oder ā̈ = ä + ¯

9) **Äquivalente Sequenzen:** sowohl ä als auch a + ¨ sind gleich zu behandeln

10) **Umwandelbarkeit (Convertibility):** Zeichendaten können zwischen Unicode und anderen Zeichenstandards konvertiert werden

Eigenschaften: normativ und informativ

- **Normative Eigenschaften:**
  - Unicode-konforme Implementierungen, die sie benutzen, müssen sich an die Spezifikation des Standards halten
  - z.B. Rendering von Arabisch oder Hebräisch
  - z.B. Groß- und Kleinschreibung

- **Informative Eigenschaften:**
  - keine präzise Information, nur als Info
  - Notizen

- **General Category** (gc) ist eine *normative* Eigenschaft und ist für alle Zeichen definiert
- gc definiert Haupt- und Unterklasse des Zeichens:

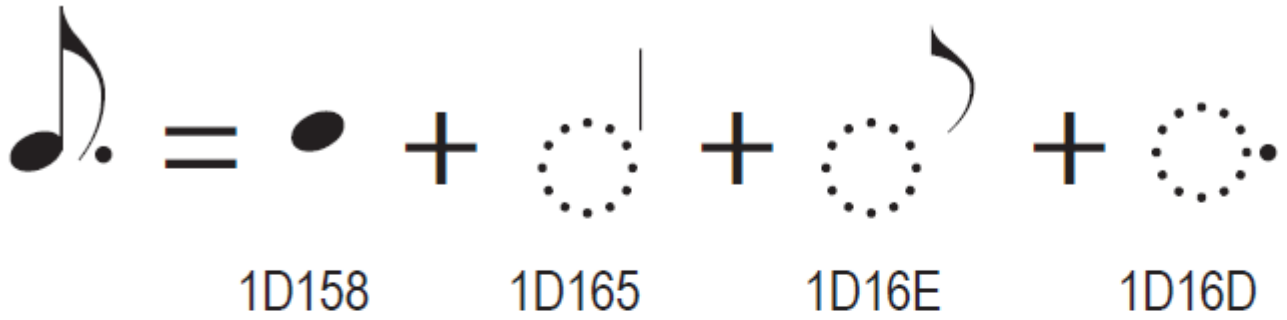
Lu = Letter, uppercase
Ll = Letter, lowercase
Lt = Letter, titlecase
Lm = Letter, modifier
Lo = Letter, other
Mn = Mark, nonspacing
Mc = Mark, spacing combining
Me = Mark, enclosing
Nd = Number, decimal digit
Nl = Number, letter
No = Number, other
Pc = Punctuation, connector
Pd = Punctuation, dash
Ps = Punctuation, open
Pe = Punctuation, close

Pi = Punctuation, initial quote
Pf = Punctuation, final quote
Po = Punctuation, other
Sm = Symbol, math
Sc = Symbol, currency
Sk = Symbol, modifier
So = Symbol, other
Zs = Separator, space
Zl = Separator, line
Zp = Separator, paragraph
Cc = Other, control
Cf = Other, format
Cs = Other, surrogate
Co = Other, private use
Cn = Other, not assigned (incl. nonchars)

flēißigę ♪

Zeichen	*General Category Wert*
fl LATIN SMALL LIGATURE FL	Ll (Letter, lowercase)
ē LATIN SMALL LETTER E WITH MACRON	Ll (Letter, lowercase)
i LATIN SMALL LETTER I	Ll (Letter, lowercase)
ß LATIN SMALL LETTER SHARP S	Ll (Letter, lowercase)
g LATIN SMALL LETTER G	Ll (Letter, lowercase)
ę LATIN SMALL LETTER E WITH OGONEK	Ll (Letter, lowercase)
SPACE	Zs (Separator, space)
♪ EIGHTH NOTE	So (Symbol, other)

Ein Basiszeichen kann mit Marks kombiniert werden



Kompositionen und Dekompositionen

- Zeichen ä kann man als ä oder als a + ¨repräsentieren

00E4 ä LATIN SMALL LETTER A WITH DIAERESIS
≡ 0061 a 0308 ◌̈

- ä (U+00E4) hat eine kanonische Dekomposition auf Zeichen a (U+0061) und ¨(U+0308)

- Es gibt also vorgefertigte (precomposed) und zerlegte (decomposed) Zeichenformen und entsprechende Konvertierungen

Kanonische und Kompatibilitätsmappings

- Dekompositionsmapping gibt es für alle Zeichen (normative E.)
- Kanonisches Mapping ≡ heißt „identisch mit“. Andere Wege, das selbe Zeichen auszudrücken. Nicht symetrisch!

0113 ē LATIN SMALL LETTER E WITH MACRON

- Latvian, Latin, ...

≡ 0065 e 0304 ◌̄

2126 Ω OHM SIGN

≡ 03A9 Ω greek capital letter omega

Kanonische und Kompatibilitätsmappings

- Kompatibilitätsmapping $\approx$ heißt „fast gleich wie“. Im Wesentlichen ähnliche Zeichen, können aber evtl. anders dargestellt werden und haben üblicherweise andere Bedeutung

FB02 ﬂ LATIN SMALL LIGATURE FL
≈ 0066 f 006C l

00B5 µ MICRO SIGN
≈ 03BC µ greek small letter mu

FF04 ＄ FULLWIDTH DOLLAR SIGN
≈ <wide> 0024 $

Kompositionen und Dekompositionen

- Historisch bedingt gibt „festgebackene“, nicht weiter zerlegbare Formen
  - norwegisches und dänisches ø
    (U+00F8) „LATIN SMALL LETTER O WITH STROKE“
  - norwegisches und dänisches æ
    (U+00E6) „LATIN SMALL LETTER AE“
  - polnisches Ł
    (U+0141) „LATIN CAPITAL LETTER L WITH STROKE“

ç

- Unicode definiert 4 Normalisierungsnormen:
  - Normalisierungsform D
    Gemäß ≡ zerlegen
  - Normalisierungsform C
    Gemäß ≡ zerlegen und dann „schlau“ zusammenbauen
  - Normalisierungsform KD
    Gemäß ≡ und ≈ zerlegen
  - Normalisierungsform KC
    Gemäß ≡ und ≈ zerlegen, dann „schlau“ zusammenbauen

flēißigę ♪

Form	*Operationen*	*Resultat*
NFD	kanonische Dekomposition	fl e ¯ i ß i g e ˛ ● | `    ē ≡ e ¯ ę ≡ e ˛ ♪ ≡ ♩` ♩ ≡ ● |
NFC	kanonische Dekomposition + kanonische Komposition	fl ē i ß i g ę ♪
NFKD	Kompatibilitätsdekomposition	f l e ¯ i ß i g e ˛ ● | ` fl ≈ f l
NFKC	Kompatibilitätsdekomposition + kanonische Komposition	f l ē i ß i g ę ♪

Welcher Buchstabe ist groß geschrieben und welcher klein?

Þ

GROSS

þ

klein

Groß- und Kleinschreibung

- 3 Case-Typen:

Lower	a	þ	dž	ω	台
Upper	A	Þ	DŽ	Ω	台
Title	A	Þ	Dž	Ω	台

- „Simple Mappings“ (normativ) für alle 3 Case Formen
- Sonderfälle:
  - http://www.unicode.org/Public/UNIDATA/SpecialCasing.txt
  - deutsches „ß“, groß: S + S, Titel: S + s
  - Ligatur „fl“, groß: F + L, Titel: F + l

Character.*toUpperCase*('ß') → 'ß'

"Fuß".toUpperCase() → "FUSS"

Case Folding

- Case Folding: wichtig für Case-insensitive Operationen
- Es wird grundsätzlich alles auf Lowercase gemappt, bis auf Sonderfälle wie das deutsche ß

fold(T) = lower(upper(T))

- Explizit für jedes Zeichen aufgeführt:

  http://www.unicode.org/Public/UNIDATA/CaseFolding.txt

- http://www.unicode.org/charts/case/

flēißigę ♪

Lowercase	fl ē i ß i g ę ♪
Uppercase	F L Ē I S S I G Ę ♪
Titlecase	F l ē i ß i g ę ♪
Folded	f l ē i s s i g ę ♪

Sortierung

Englisch	*Spanisch*
chalina	curioso
curioso	chalina
llama	luz
luz	llama

- Sortierung nach Zeichen: lexikographische Sortierung
- Oft ist die Reihenfolge der isolierten Zeichen nicht genug
- Locale-abhängig!

Vergleich und Sortierung: UCA

- Unicode Collation Algorithm (UCA) arbeitet standardmäßig mit 3 Ebenen:
  - alphabetische Reihenfolge, z.B. „a“ < „b“
  - diakritische Reihenfolge, z.B. „a“ < „á“ < „à”
  - Casing Reihenfolge, z.B. „a“ < „A“
- Kanonisch äquivalente Sequenzen gleich behandelt! Alle Input Strings mit NFD normalisiert (Zerlegung nach ≡)
- Formell: UCA(string,collationElementTable) → sortKey
- Sort Key ist eine Sequenz von 16-bit Integern (Gewichte), die eine Position des Input Strings in der Vergleichs-reihenfolge beschreiben

Ebenen: **alphabetisch**, **diakritisch**, **casing**

cote (NFD = cote)
U+0063,U+006F,U+0074,U+0065
[06EB 07F9 0869 0713 | 0020 0020 0020 0020 | 0002 0002 0002 0002]

COTE (NFD = COTE)
U+0043,U+004F,U+0054,U+0045
[06EB 07F9 0869 0713 | 0020 0020 0020 0020 | 0008 0008 0008 0008]

côte (NFD = co^te)
U+0063,U+006F,U+0302,U+0074,U+0065
[06EB 07F9 0869 0713 | 0020 0020 003C 0020 0020 | 0002 0002 0002 0002 0002]

côté (NFD = co^te´)
U+0063,U+006F,U+0302,U+0074,U+0065,U+0301
[06EB 07F9 0869 0713 | 0020 0020 003C 0020 0020 0032 | 0002 0002 0002 0002 0002 0002]

Arabischer Name für Marokko ist المغرب (maghreb)

- **Bidirektionalität:**
  - Mischung von RTL und LTR Texten, z.B. arabische Wörter in englischen Texten oder umgekehrt
  - Probleme mit Satzzeichen, die sowohl in RTL als auch in LTR benutzt werden (z.B. FULL STOP U+002E)
  - Behandlung solcher Probleme ist beschrieben in Unicode Standard Annex #9 „The Bidirectional Algorithm“ http://www.unicode.org/reports/tr9/

Arabischer Name für Marokko ist المغرب (maghreb)

- Möglichkeiten für Deklaration der Schreibrichtung:
  - Text ausschließlich LRT oder RTL deklariert
  - **UNICODE:** die Schreibrichtung wird durch unsichtbare Kontrollzeichen angegeben. Im einfachsten Fall bedeutet ein Zeichen, dass die nachfolgenden Zeichen LTR sind und eins für RTL
  - **UNICODE:** die Schreibrichtung wird inhärent der Zeichen-definition zugewiesen

- Bidi Mirroring: viele Zeichen können sowohl im LTR als auch im RTL Text auftauchen → anderer Glyph

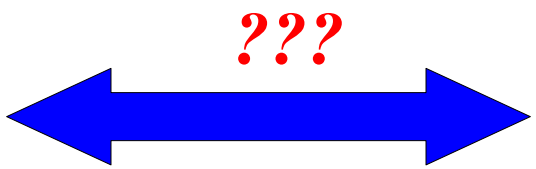
כ>א

1. א HEBREW LETTER ALEF (U+05D0)
2. > GREATER-THAN SIGN (U+003E)
3. כ HEBREW LETTER BET (U+05D1)

- http://www.unicode.org/Public/UNIDATA/BidiMirroring.txt

```
003C; 003E # LESS-THAN SIGN
003E; 003C # GREATER-THAN SIGN
007B; 007D # LEFT CURLY BRACKET
007D; 007B # RIGHT CURLY BRACKET
```

- Einführung
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- Struktur
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- Eigenschaften und Operationen
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- **Encodings**
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- Unicode in Java
- Weitere Informationsquellen

Unicode-String 

byte [ ]

- **Unicode Transition Formats (UTF)**
- Von Unicode definierte Encodings, decken ALLE Codepoints ab
- Kodierung von Codepoints auf Codeunits von *N* Bytes
- Codepoints: U+0000 bis U+10FFFF (17 Planes je $FFFF_{16}$)
- Gesamter Unicode Codespace ist mit 21 Bits kodierbar
- In der Praxis werden folgende Unicode-Kodierungen am meisten verwendet:
  - UTF-8
  - UTF-16
  - UTF-32

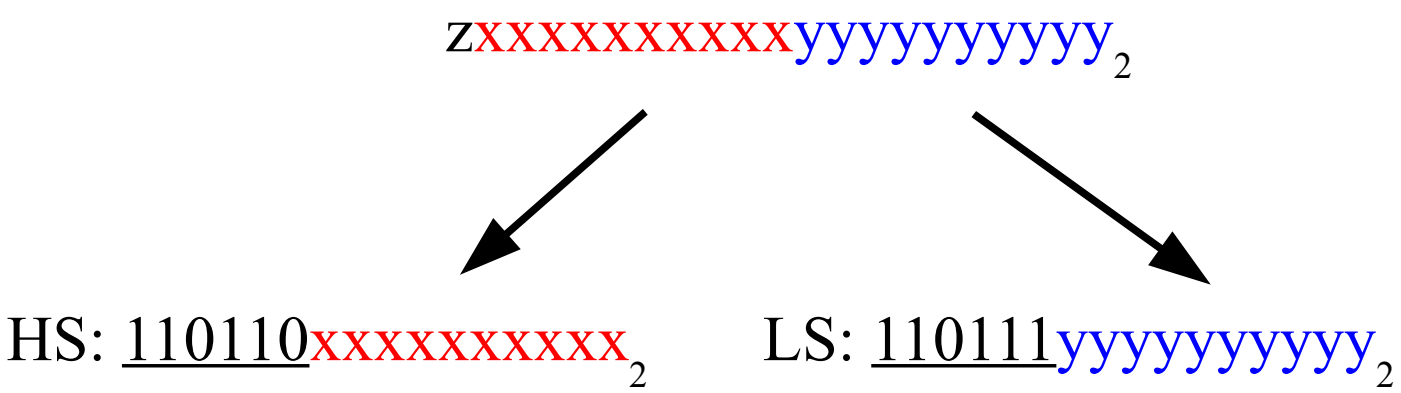
Unicode Encodings: UTF-32

- Die einfachste Kodierung, benutzt 32-Bit (4 Byte) Codeunits
- 1 Codeunit (4 Bytes) pro Zeichen

```
U+FB02   fl      [00 00 FB 02]
U+0113   ē       [00 00 01 13]
U+0069   i       [00 00 00 69]
U+00DF   ß       [00 00 00 DF]
U+0069   i       [00 00 00 69]
U+0067   g       [00 00 00 67]
U+0119   ę       [00 00 01 19]
U+0020           [00 00 00 20]
U+1D160  ♪       [00 01 D1 60]
```

Unicode Encodings: UTF-16

- 16-Bit Codeunits
- Codepoints aus BMP werden direkt kodiert (1 CP = 1 CU)
- Zeichen außerhalb BMP werden mit Surrogates auf 2 CUs kodiert
  - High Surrogates: D800 – DBFF
  - Low Surrogates: DC00 – DFFF
- Surrogates sind Codeunits, nicht Codepoints!



- UTF-16 Kodierung von U+1D160 ♪

1. Subtrahiere 10000_{16} von dem 21-Bit Wert → 20-Bit Wert

$$1D160_{16} - 10000_{16} = D160_{16} = 0000110100010110000_{2}$$

2. Man teilt die 20 Bit auf 2 x 10 Bit

$$H = 0000110100_{2} = 0034_{16}$$

$$L = 0101100000_{2} = 0160_{16}$$

3. Zum ersten Wert wird $D800_{16}$ addiert, zum zweiten $DC00_{16}$

$$HS = D800_{16} + 0034_{16} = D834_{16}$$

$$LS = DC00_{16} + 0160_{16} = DD60_{16}$$

4. UTF-16 Codeunits: [D8 34] [DD 60]

Unicode Encodings. UTF-16

```
U+FB02   fl      [FB 02]
U+0113   ē       [01 13]
U+0069   i       [00 69]
U+00DF   ß       [00 DF]
U+0069   i       [00 69]
U+0067   g       [00 67]
U+0119   ę       [01 19]
U+0020           [00 20]
U+1D160  ♪       [D8 34] [DD 60]
```

- UCS-2 kann nur BMP kodieren, dort identisch mit UTF-16

Unicode Encodings: UTF-8

- 8-Bit Codeunits
- Codepoints werden mit 1, 2, 3 oder 4 Codeunits kodiert
- Basic Latin Zeichen (ASCII, U+0000 bis U+007F) = 1 CU

Codeposition binär	*Byte 1*	*Byte 2*	*Byte 3*	*Byte 4*
00000000 0xxxxxxx 0000 – 007F	0xxxxxxx			
00000yyy yyxxxxxx 0080 – 07FF	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx 0800 – FFFF	1110zzzz	10yyyyyy	10xxxxxx	
uuuww zzzzyyyy yyxxxxxx 10000 – 10FFFF	11110uuu	10wwzzzz	10yyyyyy	10xxxxxx

Unicode Encodings: UTF-8

U+FB02	ﬂ	[EF]	[AC]	[82]	
U+0113	ē	[C4]	[93]		
U+0069	i	[69]			
U+00DF	ß	[C3]	[9F]		
U+0069	i	[69]			
U+0067	g	[67]			
U+0119	ę	[C4]	[99]		
U+0020		[20]			
U+1D160	𝅘𝅥𝅯	[F0]	[9D]	[85]	[A0]

Unicode Encodings: Byte Order

- Codeunits mit 2 oder 4 Bytes → Byte Order

  Big Endian: most significant byte first

  Little Endian: least significant byte first

- Endianness bei Datenaustausch:

  A) Byte Order von Empfänger „erraten“

  B) Byte Order explizit angegeben (in HTTP Header, etc.)

  - UTF-16LE, UTF-16BE, UTF-16 (big endian default)
  - UTF-32LE, UTF-32BE, UTF-32 (big endian default)

  C) Byte Order Mark (BOM) in den Daten

Unicode Encodings: Byte Order

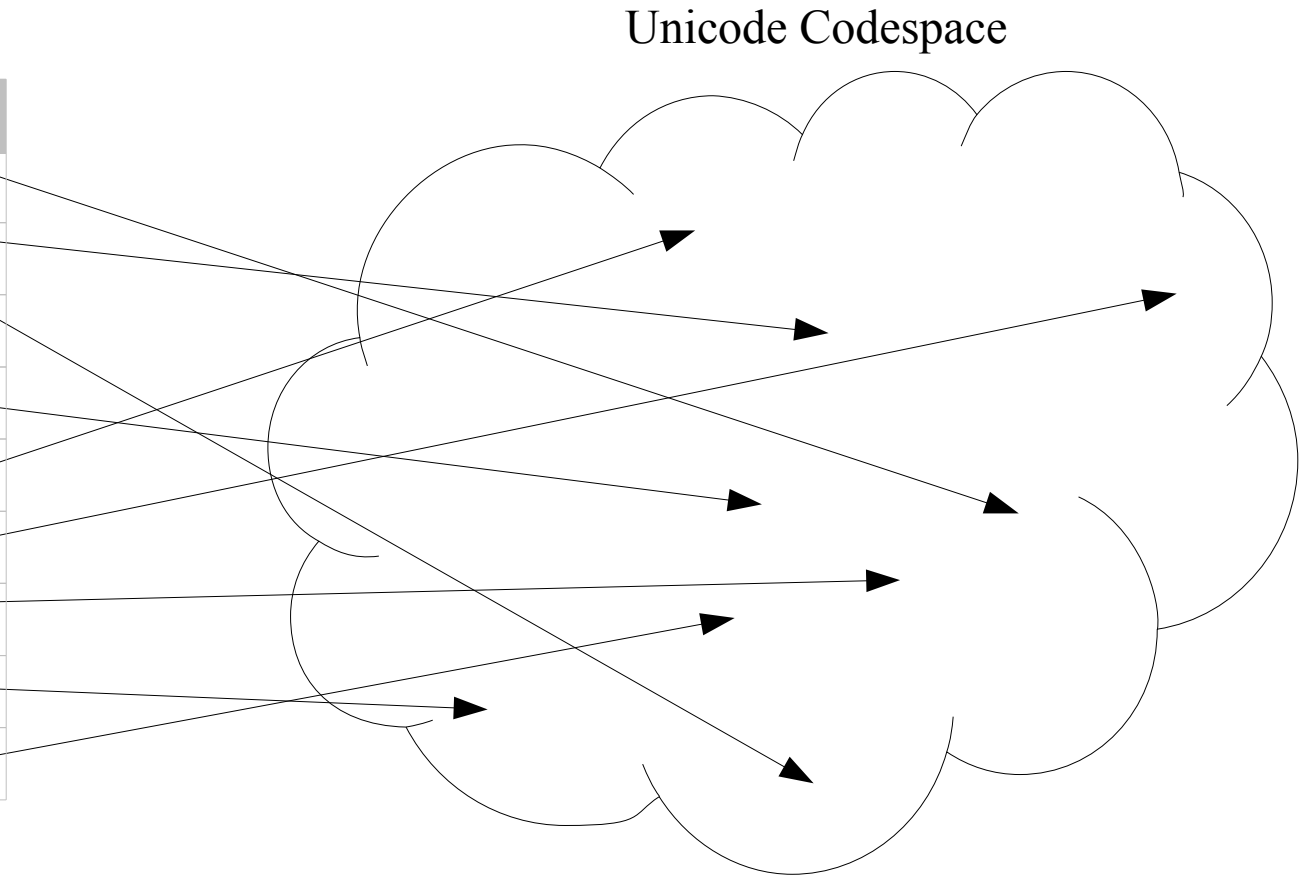
- Byte Order Mark (BOM): Codepoint U+FEFF
- Datei oder Bytestream können mit BOM anfangen
- Der selbe Codepoint! Andere Codeunits:
  - UTF-32BE [00 00 FE FF]
  - UTF-32LE [FF FE 00 00]
  - UTF-16BE [FE FF]
  - UTF-16LE [FF FE]
  - UTF-8 [EF] [BB] [BF]
- Anhand der kodierten BOM kann man das Encoding bestimmen, wenn es nicht bekannt ist

Legacy 8-Bit Encodings

- 8-Bit Encodings: nur 256 der über 1 Million Unicode Zeichen
- 1 Zeichen als 1 Byte kodiert, Bytes sind trotzdem nicht Zeichen!!!

Beispiel Windows-1252

Pos	Zeichen	Unicode
97	a	U+0061
98	b	U+0062
99	c	U+0063
138	Š	U+0160
140	Œ	U+0152
153	™	U+2122
186	º	U+00BA
192	Ã	U+00C3
241	ñ	U+00F1



Legacy 8-Bit Encodings: ISO 8859

- ISO/IEC 8859-*N*:
  - 0-127: identisch mit ASCII
  - 128-255: spezifisch
- Latin-1 bildet die ersten 255 Codepoints des Unicode-Namespace

N	Zielsprachen	Synonym
-1	Westeuropäisch	Latin-1
-2	Mitteleuropäisch	Latin-2
-3	Südeuropäisch	Latin-3
-4	Nordeuropäisch	Latin-4
-5	Kyrillisch	
-6	Arabisch	
-7	Griechisch	
-8	Hebräisch	
-9	Türkisch	Latin-5
-10	Nordisch	Latin-6
-11	Thai	
-13	Baltisch	Latin-7
-14	Keltisch	Latin-8
-15	Westeuropäisch	Latin-9
-16	Südosteuropäisch	Latin-10

Legacy 8-Bit Encodings: ISO 8859-1 & -15

- ISO 8859-1 (Latin-1) und ISO 8859-15 (Latin-9) unterscheiden sich nur auf 8 Stellen

Position	0xA4	0xA6	0xA8	0xB4	0xB8	0xBC	0xBD	0xBE
8859-1	¤	¦	¨	´	¸	¼	½	¾
8859-15	€	Š	š	Ž	ž	Œ	œ	Ÿ

- Einführung
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- Struktur
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- Eigenschaften und Operationen
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- Encodings
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- **Unicode in Java**
- Weitere Informationsquellen

- char / java.lang.Character: nur 16-Bit!
- Strings sind UTF-16 kodiert → `"flēißigę \uD834\uDD60"`
- char (0-FFFF) vs. int (Codepoint)
  - String.charAt(i) vs. String.codePointAt(i)
  - String.substring(int,int) – vielleicht Surrogates!
- String.length() vs. String.codePointCount(int,int)
  - `"flēißigę \uD834\uDD60".length()` → 10
  - `"flēißigę \uD834\uDD60".codePointCount(0,9)` → 9
- Default-Encoding für die JVM: „java -encoding UTF8 Foo“
  - *SCHLECHT:* String.getBytes()
  - *SCHLECHT:* new String(new byte[] {32,48,50})

Java: Unicode-Eigenschaften

Unicode Property	*Java Methode*
alpha (Alphabetic)	`Character.isLetterOrDigit(int)`
bc (Bidi Class)	`Character.getDirectionality(int)`
Bidi M (Midi Mirrored)	`Character.isMirrored(int)`
blk (Block)	`Character.UnicodeBlock.of(int)`
gc (General Category)	`Character.getType()`
lc (Lowercase Mapping)	`Character.toLowerCase()`
Lower (Lowercase)	`Character.isLowerCase()`
nv (Numeric Value)	`Character.getNumericValue()`
tc (Titlecase Mapping)	`Character.toTitleCase()`
uc (Uppercase Mapping)	`Character.toUpperCase()`
Upper (Uppercase)	`Character.isUpperCase()`
WSpace (White Space)	`Character.isWhiteSpace()`

Kodierung von Java-Zeichen auf Bytes:

```
String fleissigeAchtelnote = "flēißigę \uD834\uDD60";
CharBuffer charBuffer = CharBuffer
                        .wrap(fleissigeAchtelnote);

Charset utf8 = Charset.forName("UTF8");
CharsetEncoder utf8encoder = utf8.newEncoder();
ByteBuffer bytes = utf8encoder.encode(charBuffer);
```

Dekodierung von Java-Zeichen aus Bytes:

```
byte[] bytes = readBytesFromFile();
ByteBuffer buffer = ByteBuffer.wrap(bytes);

Charset utf8 = Charset.forName("UTF8");
CharsetDecoder utf8decoder = utf8.newDecoder();
CharBuffer characters = utf8decoder.decode(buffer);
```

Hint: bytes ausgeben

```
String fleissigeAchtelnote = "flēißigę \uD834\uDD60";
CharBuffer charBuffer = CharBuffer
                        .wrap(fleissigeAchtelnote);

Charset utf8 = Charset.forName("UTF8");
CharsetEncoder utf8encoder = utf8.newEncoder();

ByteBuffer bytes = utf8encoder.encode(charBuffer);

while (bytes.hasRemaining()) {
   byte b = bytes.get();
   int decimal = b & 255;
   System.out.println(decimal);
   System.out.println(Integer.toHexString(decimal));
}
```

- Java kann außerdem:
  - Normalisierung (NFD, NFC, NFKD, NFKC) mit *java.text.Normalizer*
    - normalize(CharSequence,Normalizer.Form)
    - isNormalized(CharSequence,Normalizer.Form)
  - Unicode Bidirectional Algorithm mit *java.text.Bidi*
  - Erkennung von Textgrenzen *java.text.BreakIterator*
    - Zeichen, Wörter, Sätze, Zeilen
  - UCA mit *java.text.RuleBasedCollator*
    - Locale-dependent („de_DE“, etc.)
    - Erlaubt Tailoring mit eigenen Regeln

Iteration über Zeichen eines String:

```
String s = "flēißigę \uD834\uDD60";
int codePointCount = s.codePointCount(0, s.length());
for (int i = 0; i < codePointCount; i++) {
   int codePoint = s.codePointAt(i);
   // print
}
```

```
U+fb02   LATIN SMALL LIGATURE FL
U+0113   LATIN SMALL LETTER E WITH MACRON
U+0069   LATIN SMALL LETTER I
U+00df   LATIN SMALL LETTER SHARP S
U+0069   LATIN SMALL LETTER I
U+0067   LATIN SMALL LETTER G
U+0119   LATIN SMALL LETTER E WITH OGONEK
U+0020   SPACE
U+1d160  MUSICAL SYMBOL EIGHTH NOTE
```

Normalisierung eines String:

```
String s = "flēißigę \uD834\uDD60";
String norm = Normalizer.normalize(s, Normalizer.Form.NFD);
int codePointCount = norm.codePointCount(0, norm.length());
for (int i = 0; i < codePointCount; i++) {
   int codePoint = norm.codePointAt(i);
   // print
}
```

```
   U+fb02   LATIN SMALL LIGATURE FL
   U+0065   LATIN SMALL LETTER E          \
   U+0304   COMBINING MACRON               > LATIN SMALL LETTER E WITH MACRON
   U+0069   LATIN SMALL LETTER I
   U+00df   LATIN SMALL LETTER SHARP S
   U+0069   LATIN SMALL LETTER I
   U+0067   LATIN SMALL LETTER G
   U+0065   LATIN SMALL LETTER E          \
   U+0328   COMBINING OGONEK               > LATIN SMALL LETTER E WITH OGONEK
   U+0020   SPACE
   U+1d158  MUSICAL SYMBOL NOTEHEAD BLACK \
   U+1d165  MUSICAL SYMBOL COMBINING STEM  > EIGHTH NOTE
```

International Components for Unicode (ICU)

- ICU kann zusätzlich zu Java:
  - Zeichennamen (LATIN SMALL LETTER A WITH ACUTE)
  - zusätzliche Werkzeuge für Textanalyse
  - Tailoring für java.text Funktionalität
  - Zeit und Datum: zusätzliche Kalender-Typen
  - Formatierung und Parsen
- 100% Java, eigene Zeichendatenbanken in den JARs
- ICU4C: komplett in C bzw. C++
- Wrapper für viele andere Sprachen (PHP, Cobol, Python ...)

- Einführung
  - Frühe Zeichenkodierungen, Entstehung von Unicode
- Struktur
  - Codepoints, Strings, Ebenen, Blöcke, Charts
- Eigenschaften und Operationen
  - Kompositionen und Dekompositionen, Mappings, Normalisierung, Casing, Vergleich und Sortierung
- Encodings
  - Unicode: UTF-32, UTF-16, UTF-8, Endianness, BOM
  - 8 Bit: ISO-8859, Konvertierung
- Unicode in Java
- **Weitere Informationsquellen**

Weitere Informationsquellen

- www.unicode.org
  - Kapitel in PDF Format
  - Code Charts: www.unicode.org/charts
  - Unicode Character Database: www.unicode.org/ucd
  - Unihan Database: www.unicode.org/charts/unihan.html
- JavaDoc: Character, String, java.text.*
- ICU: www.icu-project.org
- Bücher: „Unicode Standard 5.0“ von Addison Wesley, „Unicode Explained“ von O'Reilly

- Unicode deckt alle bekannten Zeichen ab, also auch die von älteren Zeichensätzen
- Zeichendefinitionen mit Integer-Codepoints von U+0000 bis U+10FFFF (kodierbar mit 21 Bits)
- Bevor man ernsthaft mit Unicode-Text etwas macht, zuerst durch NFC Normalisierung jagen
- Die Kodierung sollte zwischen Erzeuger und Empfänger der Daten ***IMMER*** bekannt sein, z.B. durch
  - HTTP Header: Content-Type: text/html; charset=utf-8
  - <?xml version="1.0" encoding="UTF-8"?>
  - Handshake, ...
- Wenn im File die BOM vorhanden ist, kann man die Kodierung „erraten“. Manuell am besten mit einem Hexeditor zu prüfen

- ISO 8859-1 und ISO 8859-15 bis auf 8 Positionen identisch
  - ISO 8859-15 hat vor allem das € Zeichen
- Kodierung und Dekodierung zwischen Codepoints (Integer) und Bytes in Java mit CharsetEncoder und CharsetDecoder
- Java char reicht nur für BMP! Oberhalb FFFF werden 2 chars benötigt, kein Random Access in Java-Strings!
- Casing: String.toUpperCase() kann String-Länge ändern
- ...und ***BITTE***, klärt in eurem Projekt, mit welchem Encoding die Source-Dateien eingecheckt werden ;-)

```
/**
* @author Lieschen MÃ¼ller
*/
```

Vielen Dank!

karol.rueckschloss@mathema.de